Crash Recovery в Распределенном Хранилище

Антон Виноградов,
Apache Ignite Committer & PMC Member





Crash Recovery

Что такое **Database Crash**?



Database Crash

Exception/Error



Database Recovery (ожидания)

Exception/Error



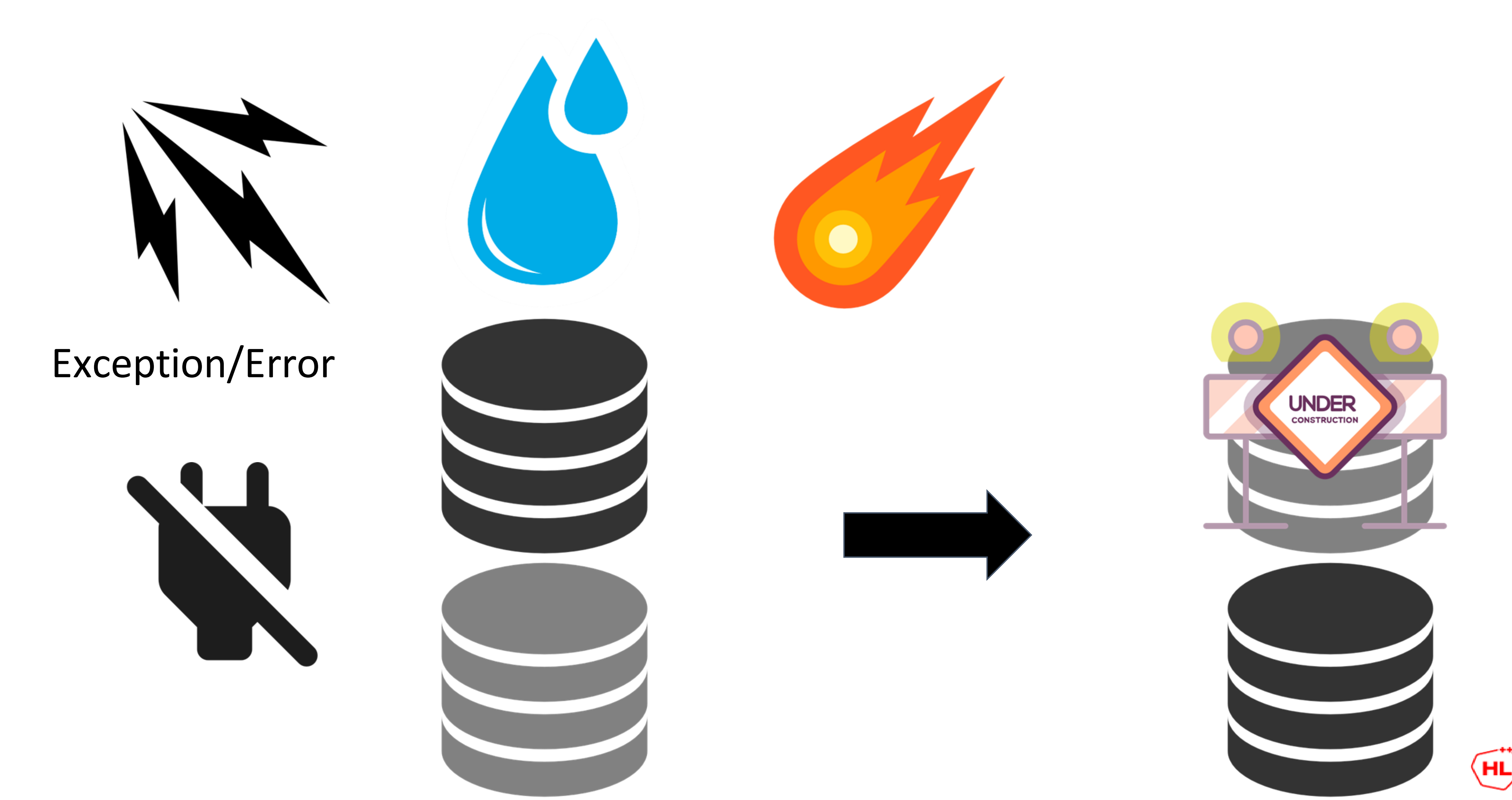
Database Recovery (реальность)

Exception/Error

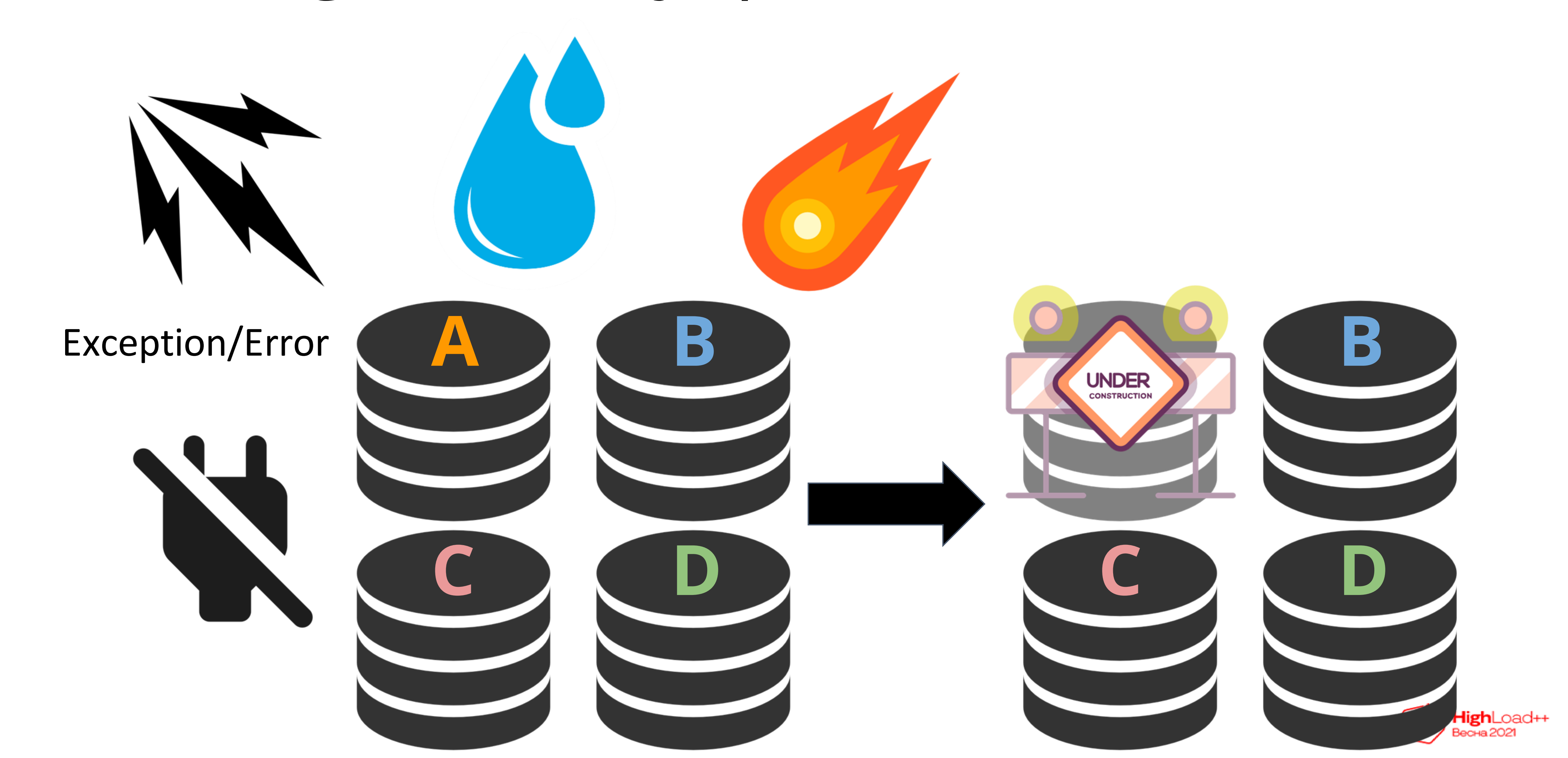
UNDER CONSTRUCTION



Master-Follower Recovery (реальность)



Sharding Recovery (реальность)



Распределенные Хранилища

A B

A B

A B

A B

A B



Crash в Распределенном Хранилище

A B

A B

A B

A B

A B



Crash Recovery (ожидания)





Crash Recovery (реальность)

A B

A B

A B

A B



Need to go deeper





Кэш (Cache)

Cache в Apache Ignite — это распределенная структура, близкая по семантике к ***ConcurrentHashMap<K,V>***.

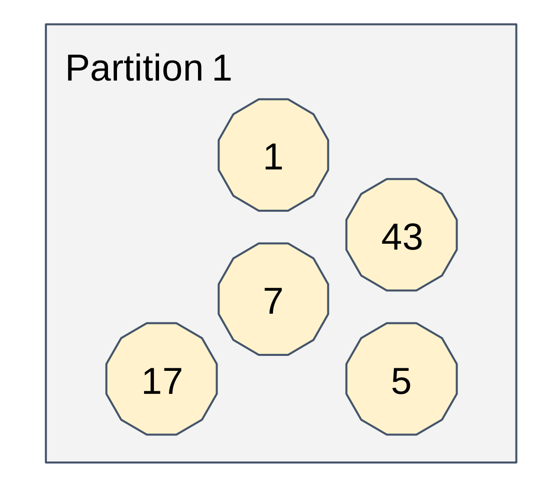
Или согласно документации:

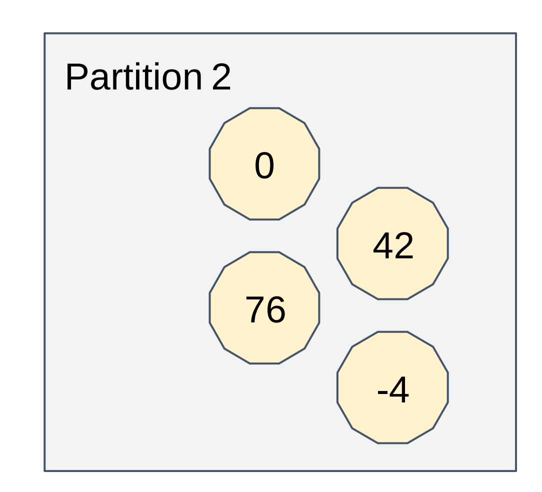
IgniteCache is based on J*Cache (**JSR 107**)*, so at the very basic level the APIs can be reduced to the ***javax.cache.Cache***.



Партиционирование данных

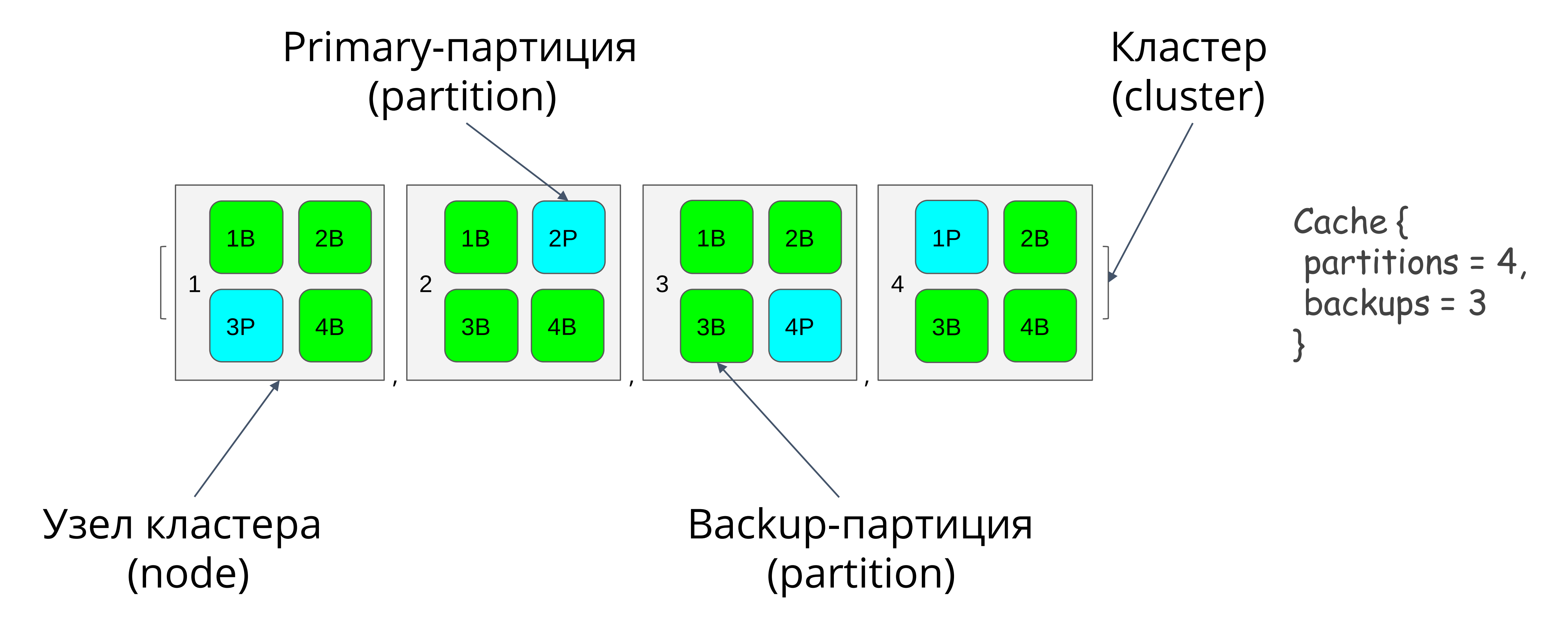
f = (key) → key.hashCode() % partitions.size();
part = f(key);







Партиция, Кэш, Кластер



Consistency

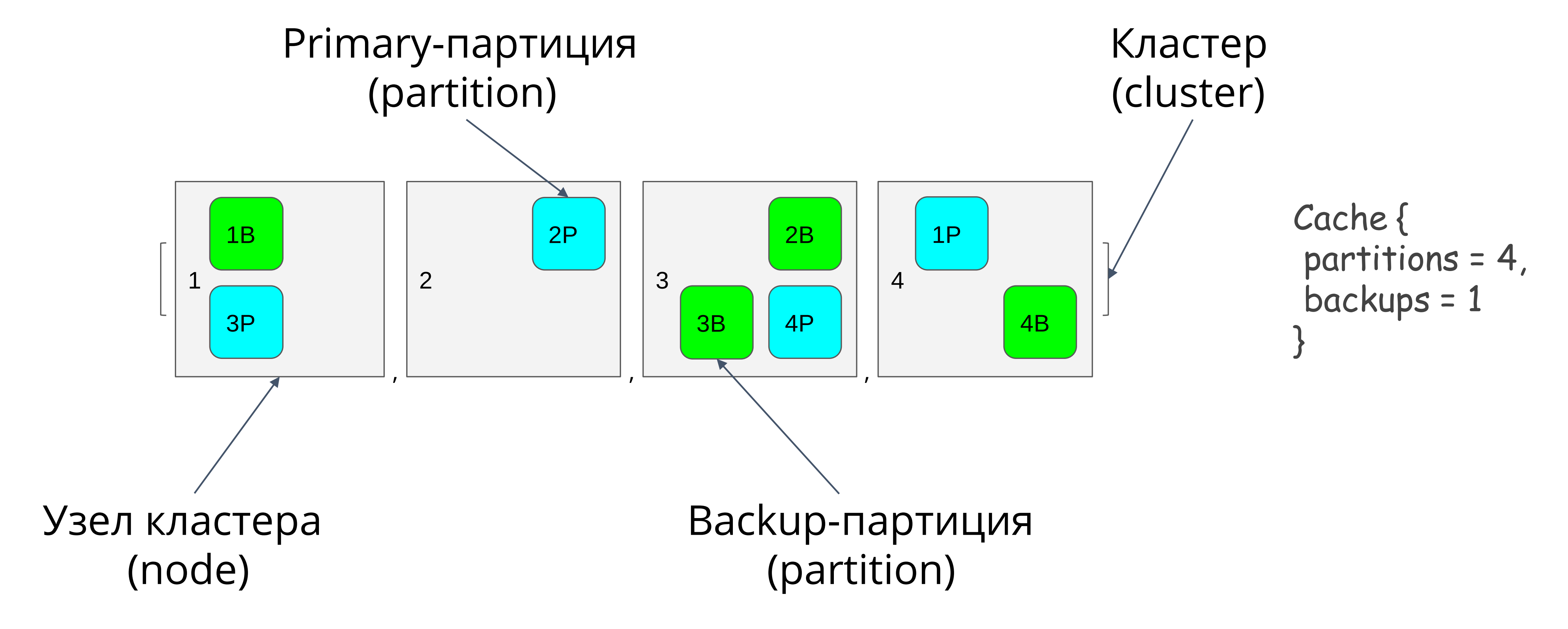
В любой конкретный момент времени, только один узел кластера должен обрабатывать обновления по конкретной партиции.

В противном случае возможны ошибки и нарушения консистентности.

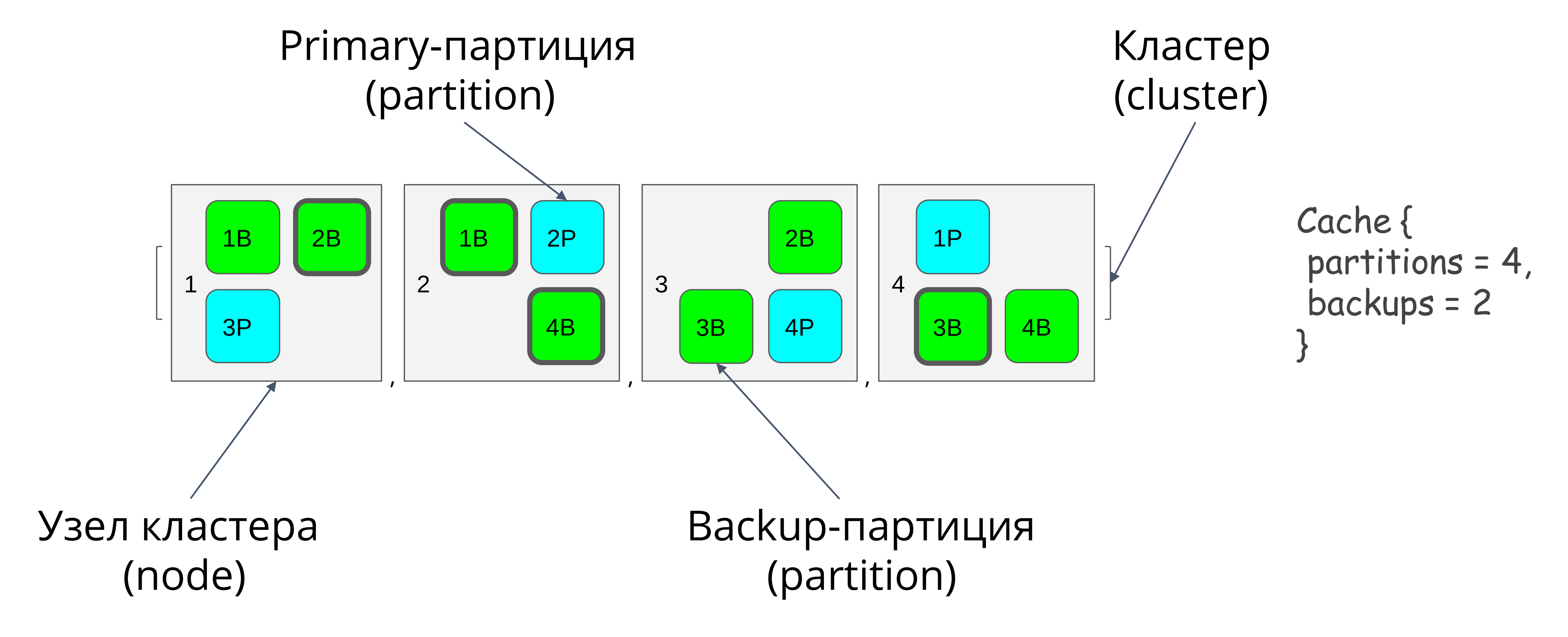




Partitioned Cache



Partitioned Cache (2 backups)



Replicated = Partitioned (∞ backups) Cache

Primary-партиция
(partition)

Кластер
(cluster)

1: 1B, 2B, 3P, 4B	2: 1B, 2P, 3B, 4B	3: 1B, 2B, 3B, 4P	4: 1P, 2B, 3B, 4B

```
Cache {
 partitions = 4,
 backups = max
}
```

Узел кластера
(node)

Backup-партиция
(partition)



Topology

Кластер
(cluster)

1 1B 2B 3P 4B , 2 1B 2P 3B 4B , 3 1B 2B 3B 4P , 4 1P 2B 3B 4B

```
Topology {
 version = 4,
 nodes = [1, 2, 3, 4]
}
```

Узел кластера
(node)



Изменение топологии: выход узла

[
1: 1B 2B 3P 4B,
2: 1B 2P 3B 4B,
3: 1B 2B 3B 4P,
4: 1P 2B 3B 4B
]

Topology {
version = 4,
nodes = [1, 2, **3**, 4]
}

[
1: 1B 2B 3P 4B,
2: 1B 2P 3B 4B,
3: 1B 2B 3B 4P
]

Topology {
version = **5**,
nodes = [1, 2, 3]
}



Изменение топологии: вход узла

1
1B 2B
3P 4B
2
1B 2P
3B 4B
3
1B 2B
3B 4P
4
1P 2B
3B 4B

Topology {
version = 4,
nodes = [1, 2, 3, 4]
}

1
1B 2B
3P 4B
2
1B 2P
3B 4B
3
1B 2B
3B 4P
4
1P 2B
3B 4B
5

Topology {
version = **5**,
nodes = [1, 2, 3, 4, **5**]
}



Switch (Переключение)

Каждая топология имеет определенную, строго инкрементальную версию.

Switch — процесс перехода с одной версии топологии на другую.



Switch

До версии 2.8 все изменения топологии проходили по единому сценарию

1) Блокируем все новые операции
2)
3)
4)
5)
6) Начинаем обрабатывать новые операции



Switch

До версии 2.8 все изменения топологии проходили по единому сценарию

1) Блокируем все новые операции
2) Восстанавливаем операции, “связанные” с вышедшими узлами
3)
4)
5)
6) Начинаем обрабатывать новые операции



Switch

До версии 2.8 все изменения топологии проходили по единому сценарию

1) Блокируем все новые операции
2) Восстанавливаем операции, “связанные” с вышедшими узлами
3) Дожидаемся завершения уже запущенных операций
4)
5)
6) Начинаем обрабатывать новые операции



Switch

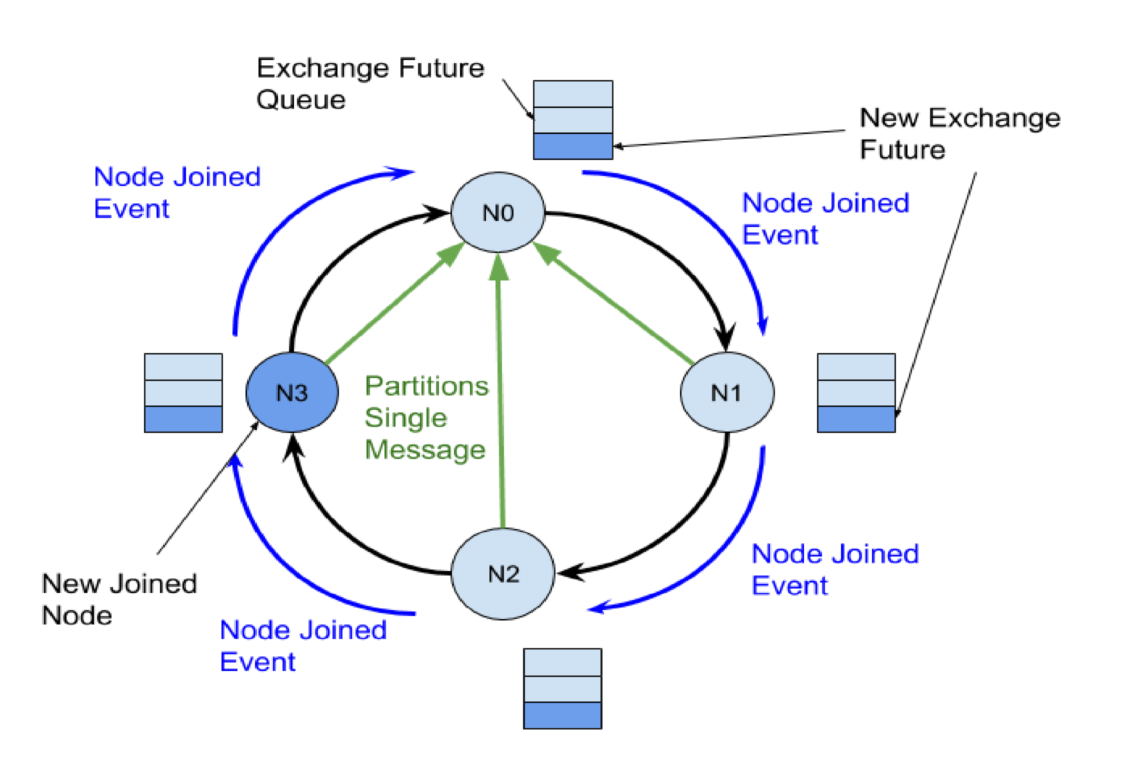
До версии 2.8 все изменения топологии проходили по единому сценарию

1) Блокируем все новые операции
2) Восстанавливаем операции, “связанные” с вышедшими узлами
3) Дожидаемся завершения уже запущенных операций
4) PME (Partition Map Exchange)
5)
6) Начинаем обрабатывать новые операции

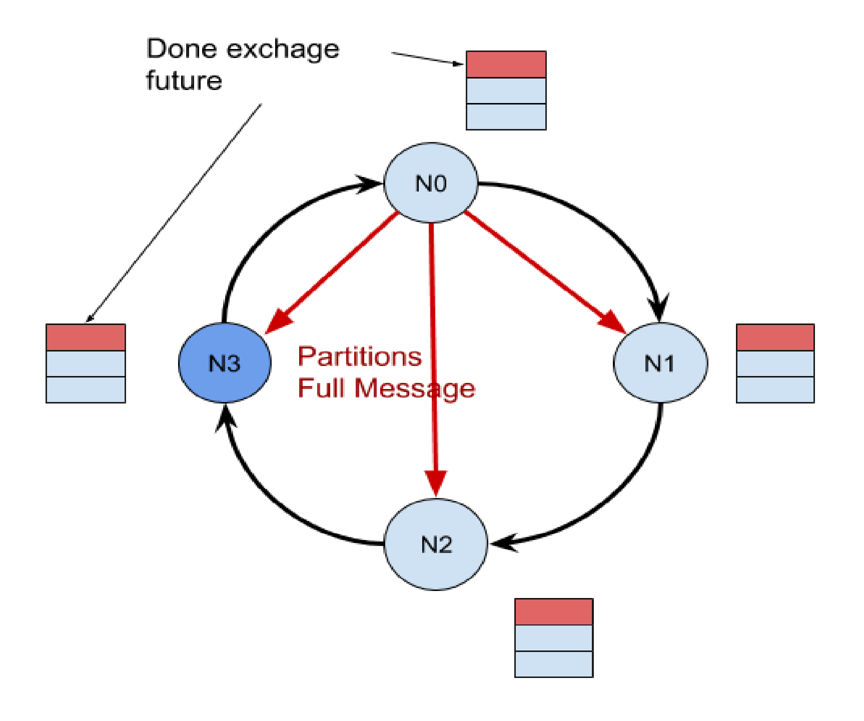


PME (Partition Map Exchange)

#1 Получаем данные о статусах партиций на узлах кластера



#2 Вычисляем распределение на новую топологию и распространяем на узлы кластера





Switch

До версии 2.8 все изменения топологии проходили по единому сценарию

1) Блокируем все новые операции
2) Восстанавливаем операции, “связанные” с вышедшими узлами
3) Дожидаемся завершения уже запущенных операций
4) PME (Partition Map Exchange)
5)
6) Начинаем обрабатывать новые операции



Switch

До версии 2.8 все изменения топологии проходили по единому сценарию

1) Блокируем все новые операции
2) Восстанавливаем операции “связанные” с вышедшими узлами
3) Дожидаемся завершения уже запущенных операций
4) PME (Partition Map Exchange)
5) Каждый узел применяет новое распределение
6) Начинаем обрабатывать новые операции



Вход узла

Affinity
1 = [1],
2 = [1],
3 = [1],
4 = [1]

1	1P	2P
	3P	4P

\+ 2

→

Affinity
partitions {
1 = [1, 2],
2 = [2, 1],
3 = [1, 2],
4 = [2, 1]
} nodes

[1: 1P, 2B, 3P, 4B ; 2: 1B, 2P, 3B, 4P]

```
Cache {
 partitions = 4,
 backups = 2
}
```




Вход узла

Affinity
1 = [1],
2 = [1],
3 = [1],
4 = [1]

1	1P	2P
	3P	4P

\+

2

→

[

1	1P	2B
	3P	4B

,

2	1B	2P
	3B	4P

]

Affinity
partitions {
1 = [1, 2],
2 = [2, 1],
3 = [1, 2],
4 = [2, 1]
} nodes

```
Cache {
 partitions = 4,
 backups = 2
}
```




Вход узла

Affinity
1 = [1],
2 = [1],
3 = [1],
4 = [1]

1		
	1P	2P
	3P	4P

\+

2

→

[

1		
	1P	2B
	3P	4B

,

2		
	1B	2P
	3B	4P

]

partitions

Affinity
1 = [1, 2],
2 = [2, 1],
3 = [1, 2],
4 = [2, 1]

nodes

Cache {
partitions = 4,
backups = 2
}



Вход узла

Affinity
1 = [1],
2 = [1],
3 = [1],
4 = [1]

1	1P	2P
	3P	4P

\+

2

→

Affinity

partitions
1 = [1, 2],
2 = [2, 1],
3 = [1, 2],
4 = [2, 1]
nodes

[

1	1P	2B
	3P	4B

,

2	1B	2P
	3B	4P

]

```
Cache {
 partitions = 4,
 backups = 2
}
```




Вход узла

Affinity
1 = [1],
2 = [1],
3 = [1],
4 = [1]

1	
1P	2P
3P	4P

\+

2

→

Affinity

partitions { 1 = [1, 2], 2 = [2, 1], 3 = [1, 2], 4 = [2, 1] } nodes

[

1	
1P	2B
3P	4B

,

2	
1B	2P
3B	4P

]

```
Cache {
 partitions = 4,
 backups = 2
}
```




Switch №1 — Join

1P 2P
1
3P 4P

2
1M 2M
3M 4M

Affinity
1 = [**1**, 2],
2 = [**2**, 1],
3 = [**1**, 2],
4 = [**2**, 1]

[
1
1P 2B
3P 4B
,
2
1B 2P
3B 4P
]

Cache {
partitions = 4,
backups = 2
}



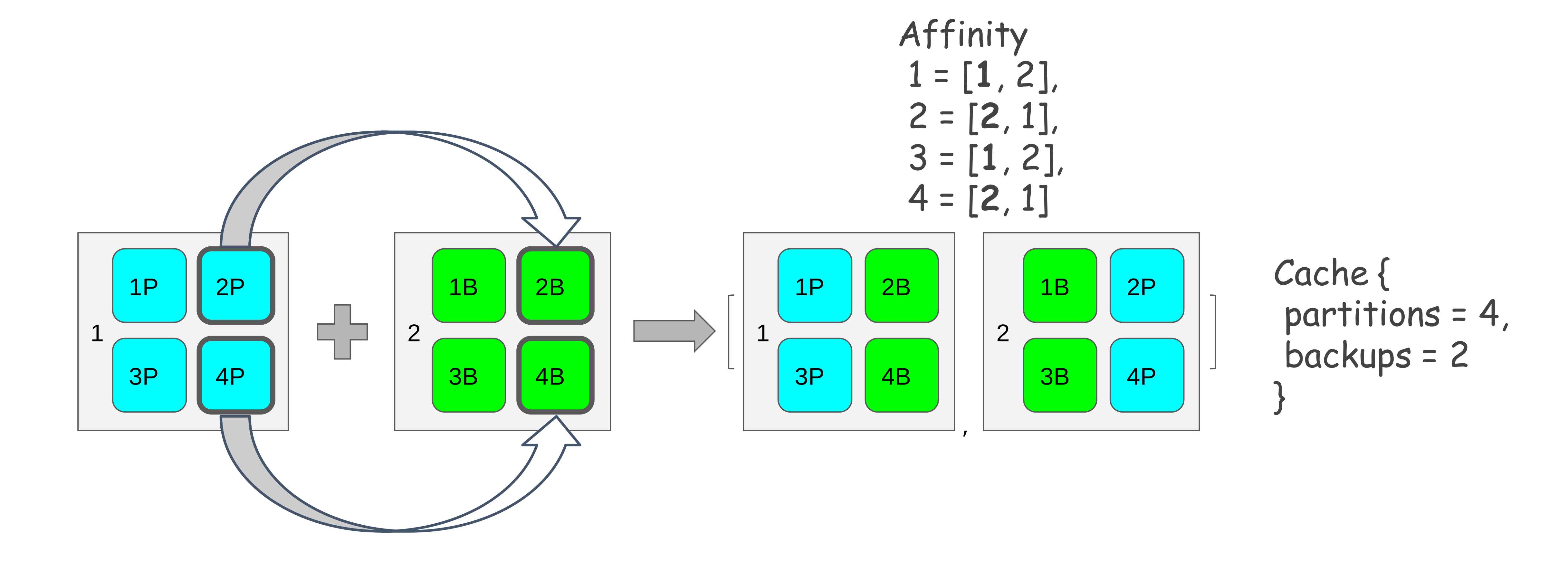
Switch №1 — Rebalanced

Affinity
1 = [**1**, 2],
2 = [**2**, 1],
3 = [**1**, 2],
4 = [**2**, 1]

Cache {
partitions = 4,
backups = 2
}



Switch №1 — Rebalanced



Switch №2 — Late Affinity Assignment

Affinity
1 = [**1**, 2],
2 = [**2**, 1],
3 = [**1**, 2],
4 = [**2**, 1]

1		
	1P	2B
	3P	4B

+

2		
	1B	2P
	3B	4P

→

[

1		
	1P	2B
	3P	4B

,

2		
	1B	2P
	3B	4P

]

```
Cache {
 partitions = 4,
 backups = 2
}
```




Выход узла (2.8)

Topology
1 = [2, 3, 1],
2 = [2, 3],
3 = [1, 2],
4 = [3, 1]

Affinity
1 = [3, 1],
2 = [1, 3],
3 = [3, 1],
4 = [1, 3]

[1: 1B, 2M, 3P, 4B ; 2: 1P, 2P, 3B, 4M ; 3: 1B, 2B, 3M, 4P] → [1: 1B, 2P, 3B, 4P ; 3: 1P, 2B, 3P, 4B]

Cache {
partitions = 4,
backups = 2
}

Выходящий
узел



Switch №1 — Left (PME)

Primary должна быть здесь

Affinity
1 = [**3**, 1],
2 = [**1**, 3],
3 = [**3**, 1],
4 = [**1**, 3]

1	1B	2M	
	3P	4B	
3	1B	2B	
	3M	4P	

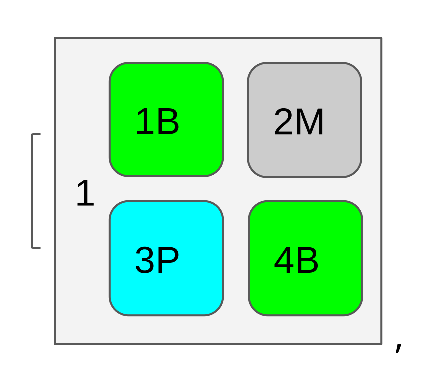
→

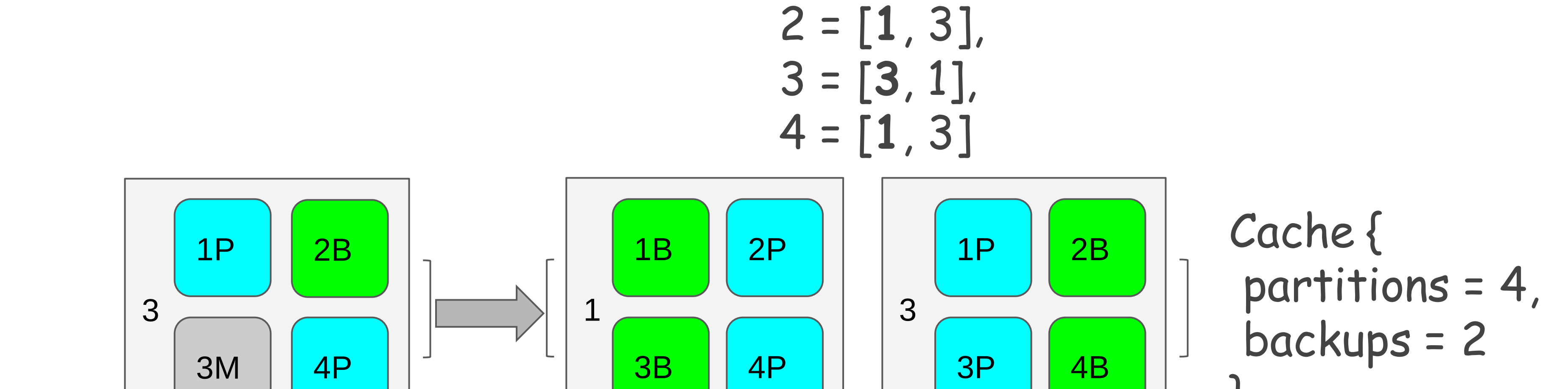
1	1B	2P
	3B	4P
3	1P	2B
	3P	4B

Cache {
partitions = 4,
backups = 2
}



Switch №1 — Left (PME)







Switch №1 — Left (PME)

Primary должна быть здесь

Но временно будет здесь

Affinity
1 = [**3**, 1],
2 = [**1**, 3],
3 = [**3**, 1],
4 = [**1**, 3]

[1: 1B, 2M, 3P, 4B , 3: 1P, 2B, 3M, 4P] → [1: 1B, 2P, 3B, 4P , 3: 1P, 2B, 3P, 4B]

Cache {
partitions = 4,
backups = 2
}



Switch №1 — Left (PME)

Primary должна быть здесь

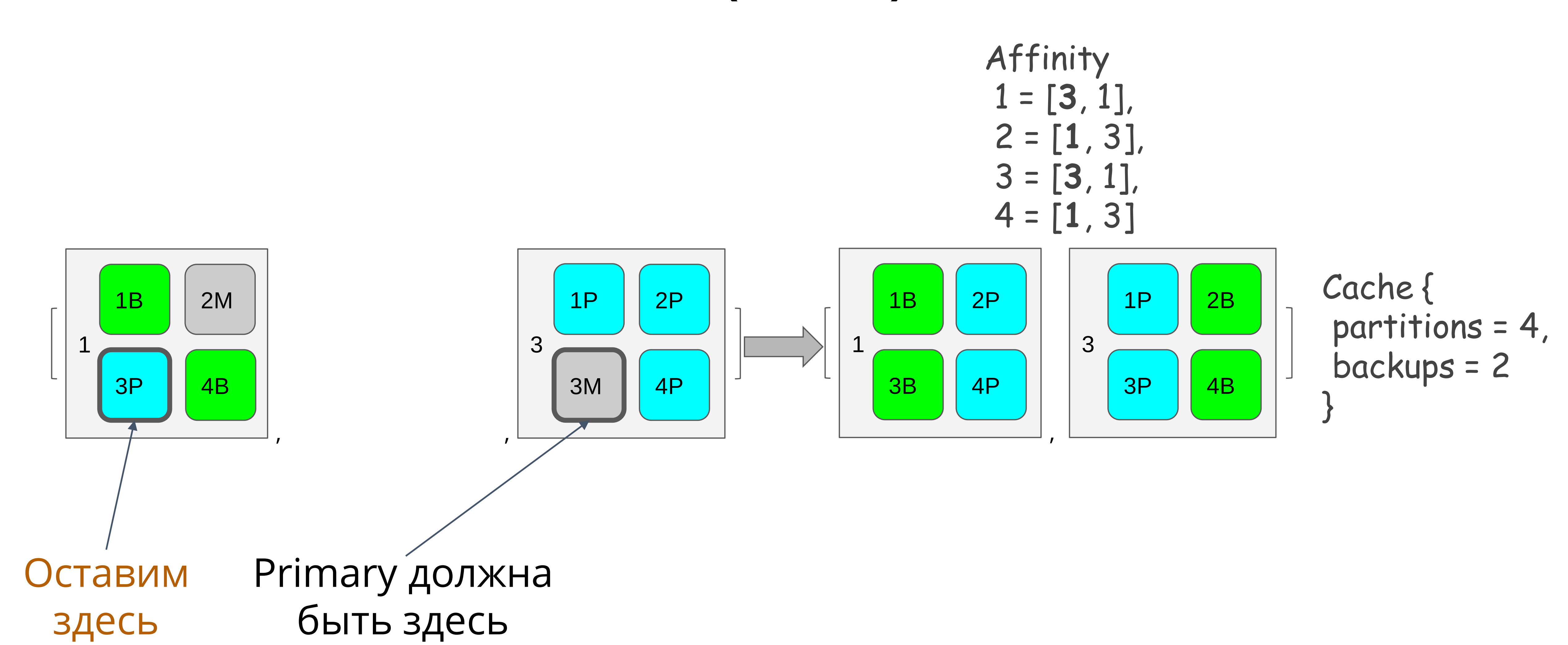
Affinity
1 = [3, 1],
2 = [1, 3],
3 = [3, 1],
4 = [1, 3]

[1: 1B 2M 3P 4B , 3: 1P 2P 3M 4P] → [1: 1B 2P 3B 4P , 3: 1P 2B 3P 4B]

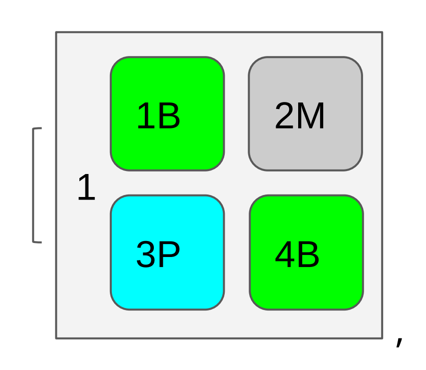
Cache {
partitions = 4,
backups = 2
}

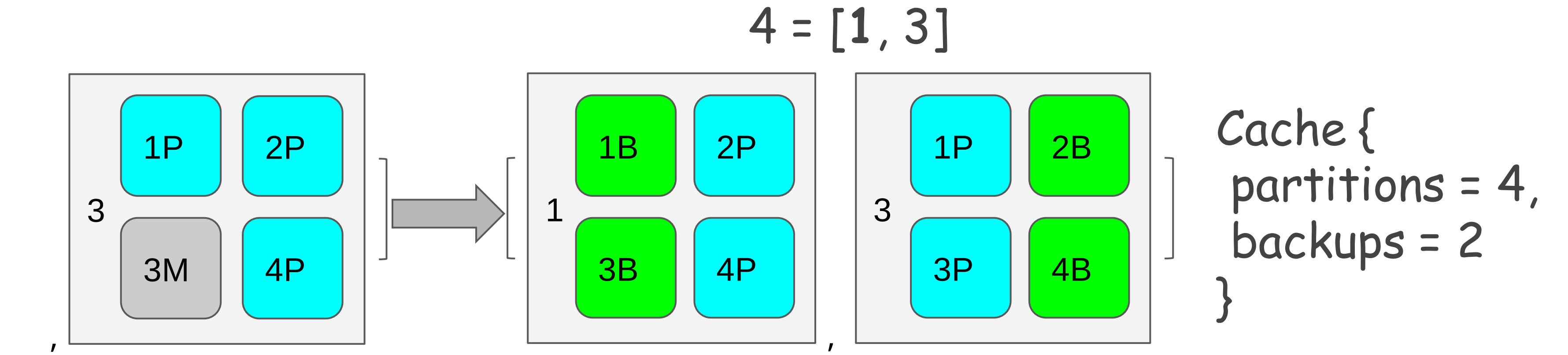


Switch №1 — Left (PME)



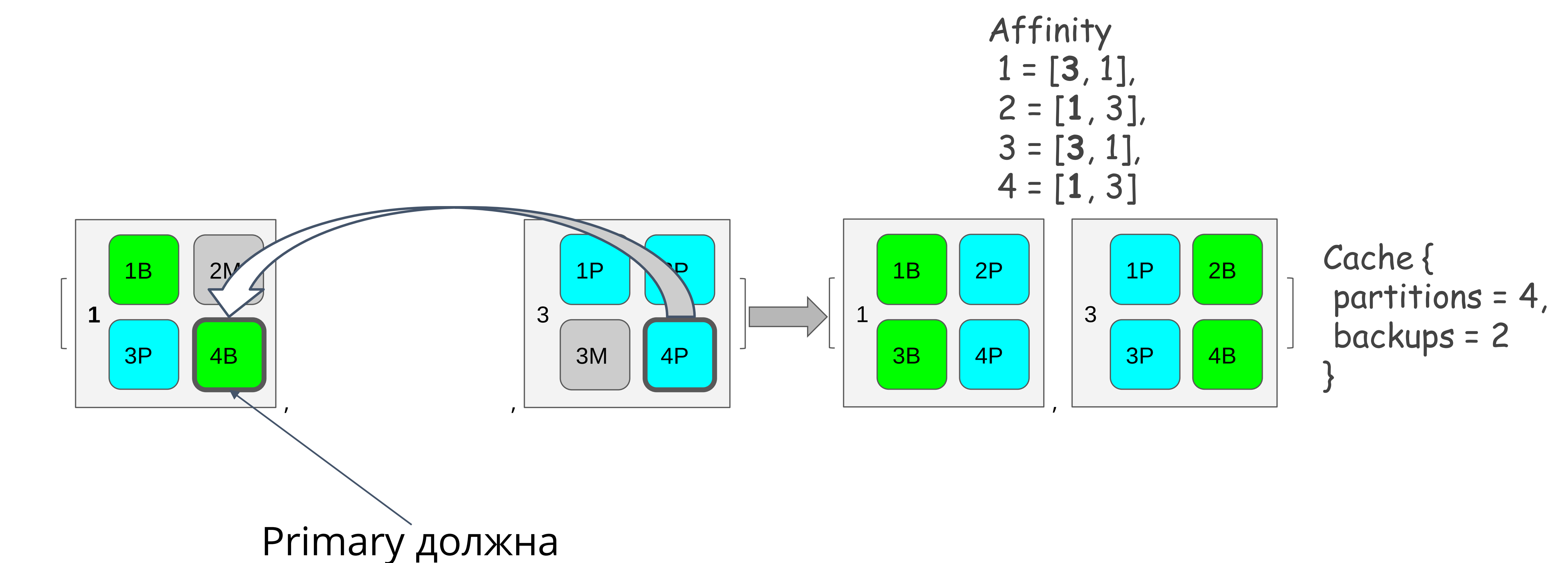
Switch №1 — Left (PME)



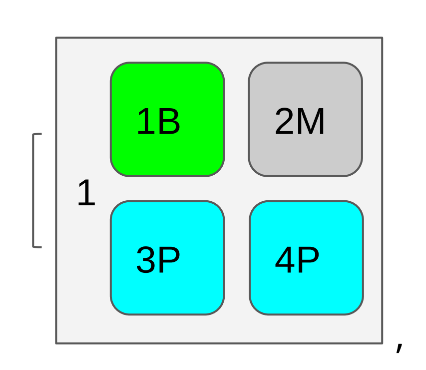


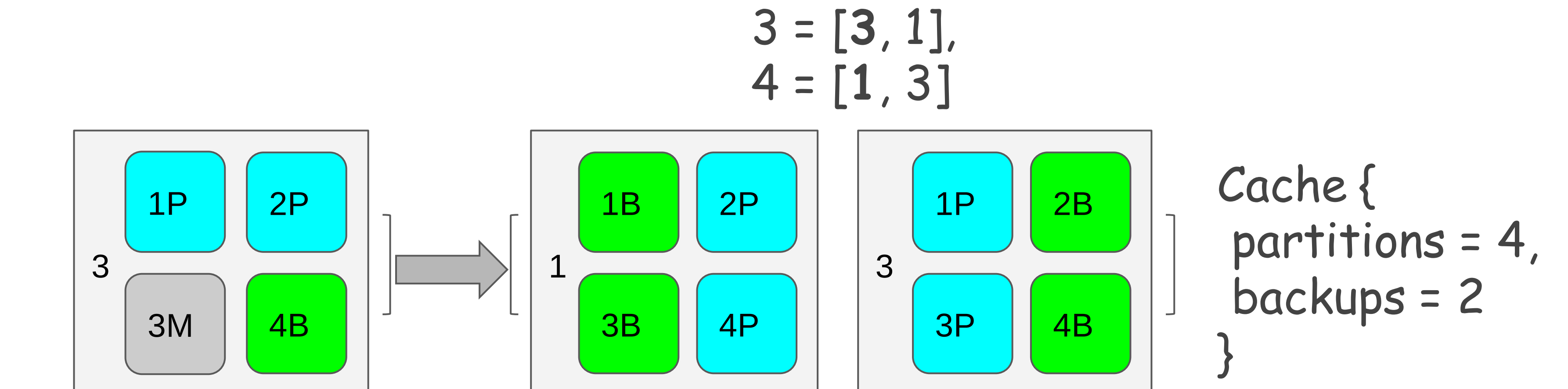


Switch №1 — Left (PME)



Switch №1 — Left (PME)

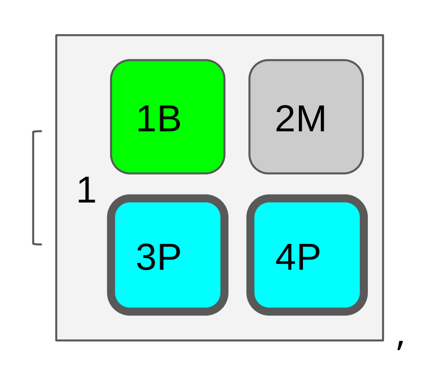


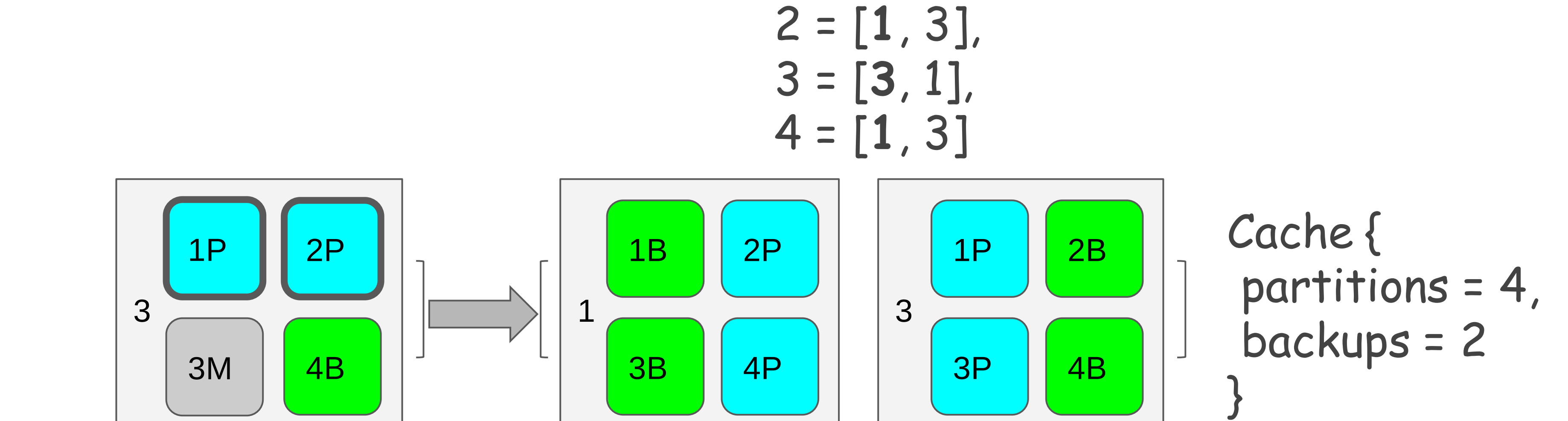


Cache {
partitions = 4,
backups = 2
}



Switch №1 — Left (PME)







Switch №1 — Rebalancing

Primary должна быть здесь

Affinity
1 = [**3**, 1],
2 = [**1**, 3],
3 = [**3**, 1],
4 = [**1**, 3]

1		
	1B	2M
	3P	4P

3		
	1P	2P
	3M	4B

→

1		
	1B	2P
	3B	4P

3		
	1P	2B
	3P	4B

Cache {
partitions = 4,
backups = **2**
}

Primary должна быть здесь



Switch №1 — Rebalancing

Primary должна быть здесь

Affinity
1 = [**3**, 1],
2 = [**1**, 3],
3 = [**3**, 1],
4 = [**1**, 3]

[1: 1B, 2M, 3P, 4P , 3: 1P, 2P, 3M, 4B] → [1: 1B, 2P, 3B, 4P , 3: 1P, 2B, 3P, 4B]

Cache {
partitions = 4,
backups = 2
}

Primary должна быть здесь



Switch №1 — Rebalanced

Primary должна быть здесь

Affinity
1 = [**3**, 1],
2 = [**1**, 3],
3 = [**3**, 1],
4 = [**1**, 3]

[1: 1B 2B 3P 4P , 3: 1P 2P 3B 4B] → [1: 1B 2P 3B 4P , 3: 1P 2B 3P 4B]

Cache {
partitions = 4,
backups = 2
}

Primary должна быть здесь



Switch №2 — Late Affinity Assignment

Primary должна быть здесь

Affinity
1 = [3, 1],
2 = [1, 3],
3 = [3, 1],
4 = [1, 3]

1	1B	2B
	3P	4P

3	1P	2P
	3B	4B

→

1	1B	2P
	3B	4P

3	1P	2B
	3P	4B

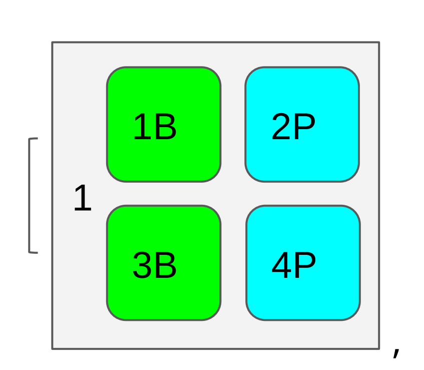
Cache {
partitions = 4,
backups = 2
}

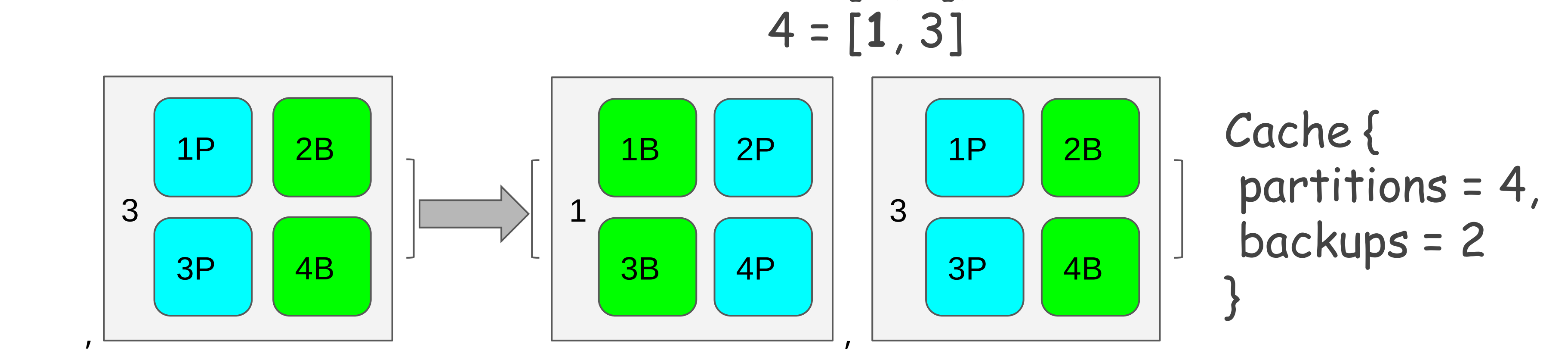
Primary должна быть здесь



Switch №2 — Late Affinity Assignment

Affinity
1 = [**3**, 1],
2 = [**1**, 3],
3 = [**3**, 1],
4 = [**1**, 3]

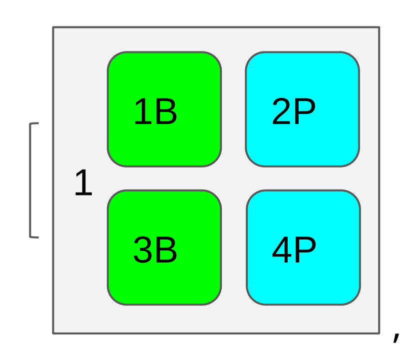


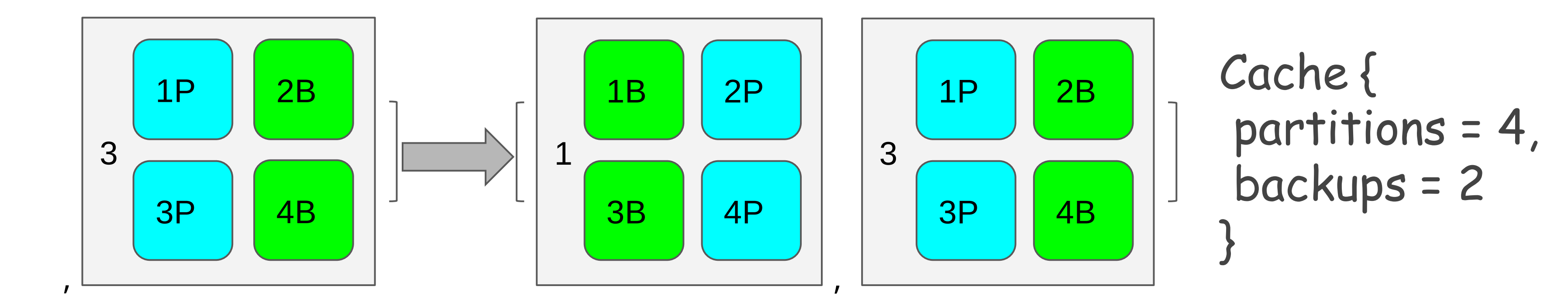




Switch №2 — Late Affinity Assignment

2 переключения :(







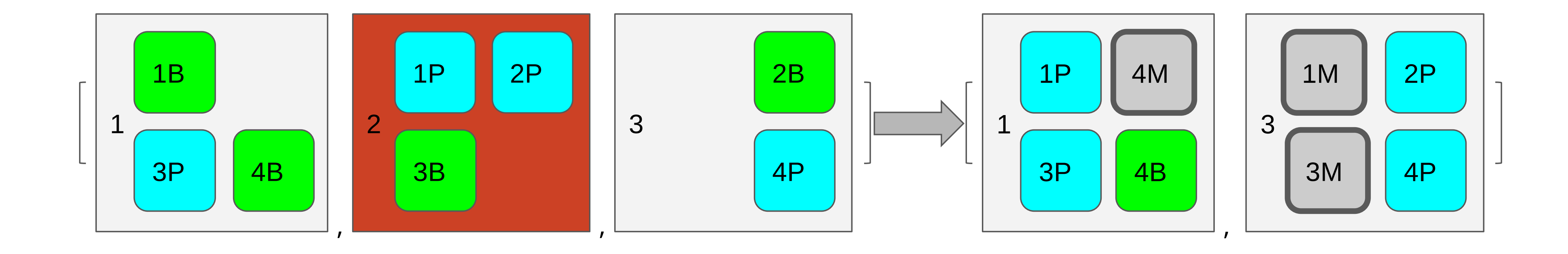
А можно восстановиться быстрее?





Выход узла

Выход узла провоцирует ребалансировку данных.

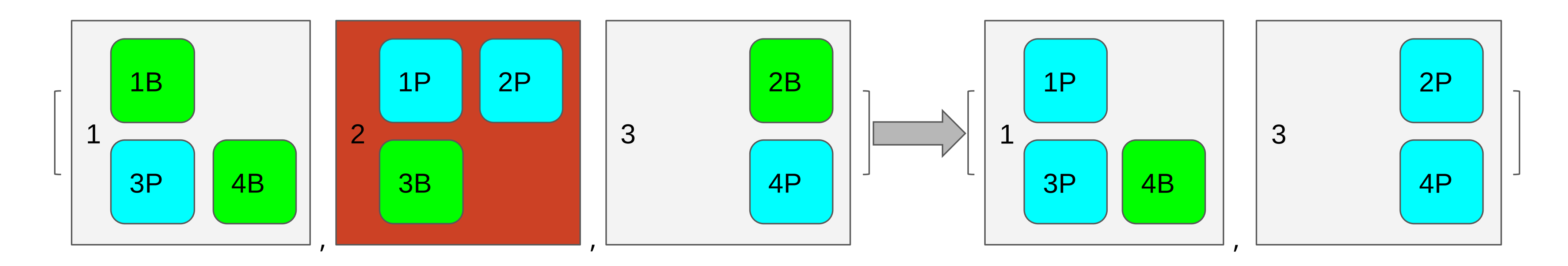




Baseline topology

Baseline topology фиксирует узлы кластера, на которых могут храниться данные.

Набор партиций на узлах тоже фиксируется.



Возможны смены статуса *Backup* в *Primary* (не наоборот).

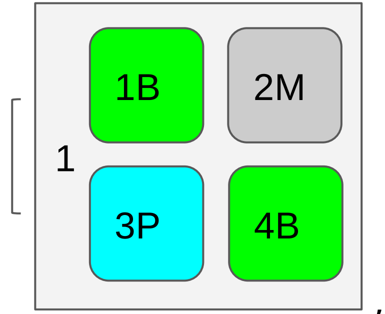
Нет последующей ребалансировки.

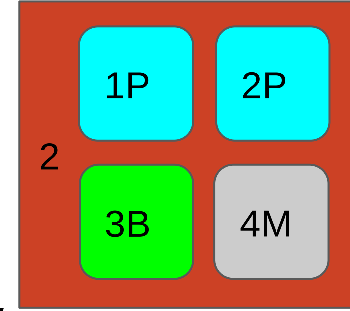


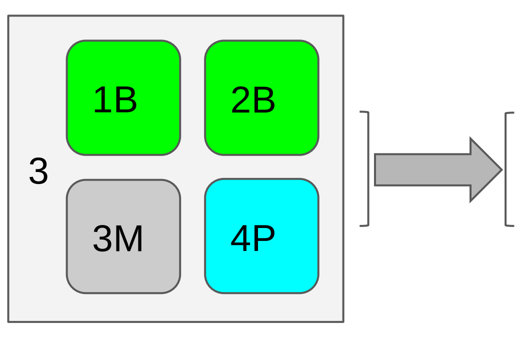
Baseline Node Left

Fixed Affinity
1 = [**2**, 1, 3],
2 = [**2**, 3, 1],
3 = [**1**, 2, 3],
4 = [**3**, 1, 2]

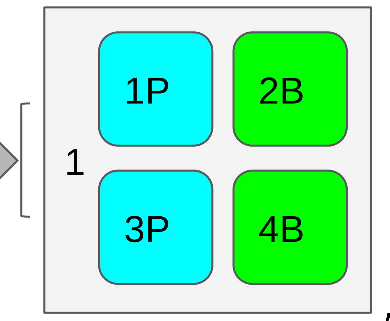
Fixed Affinity
1 = [**2**, **1**, 3],
2 = [**2**, **3**, 1],
3 = [**1**, **2**, 3],
4 = [**3**, 1, **2**]

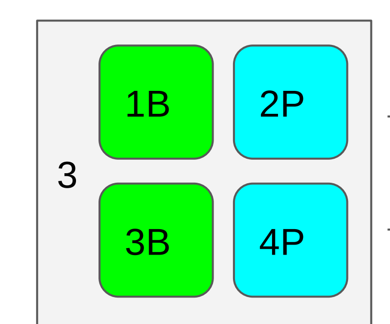
[


,


,


]

[


,


]

Cache {
 partitions = 4,
 backups = 2
}

Cluster {
 baseline = true
}



Switch №1 — Left

Primary должны быть здесь

Fixed Affinity
1 = [2, **1**, 3],
2 = [2, **3**, 1],
3 = [**1**, 2, 3],
4 = [**3**, 1, 2]

1		
	1B	2M
	3P	4B

3		
	1B	2B
	3M	4P

→

1		
	1P	2B
	3P	4B

3		
	1B	2P
	3B	4P

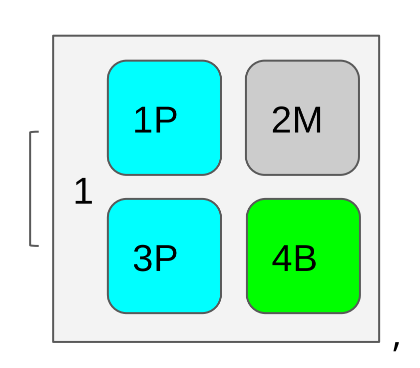
```
Cache {
 partitions = 4,
 backups = 2
}

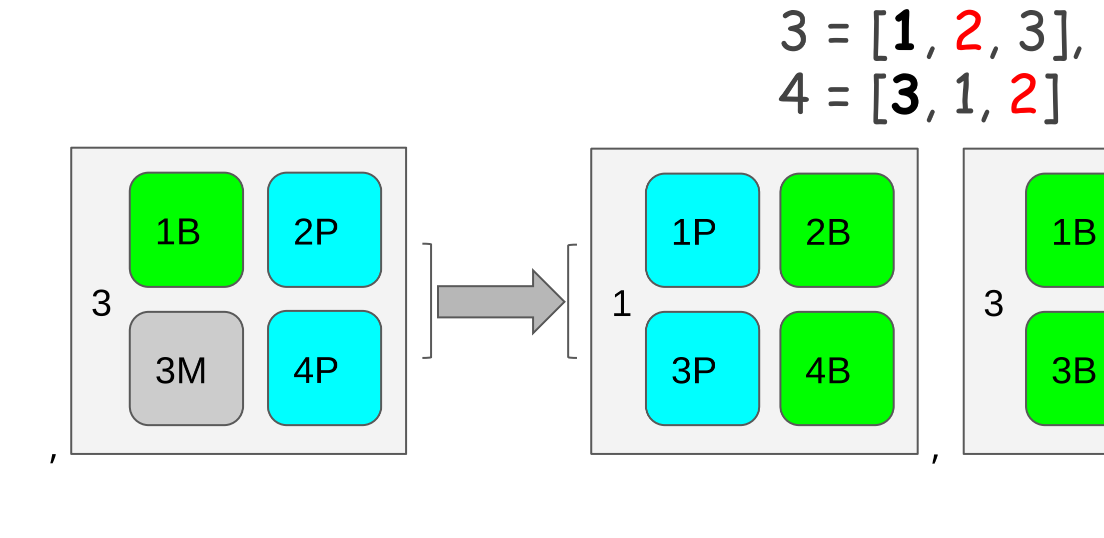
Cluster {
 baseline = true
}
```




Switch №1 — Left

Fixed Affinity
1 = [2, **1**, 3],
2 = [2, **3**, 1],
3 = [**1**, 2, 3],
4 = [**3**, 1, 2]





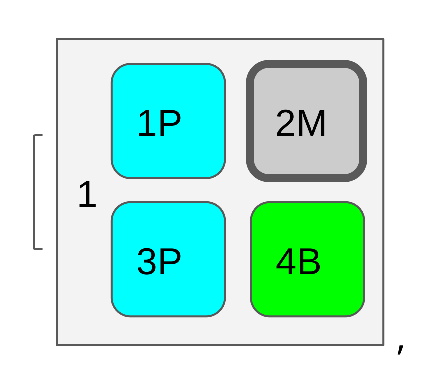
Cache {
 partitions = 4,
 backups = **2**
}

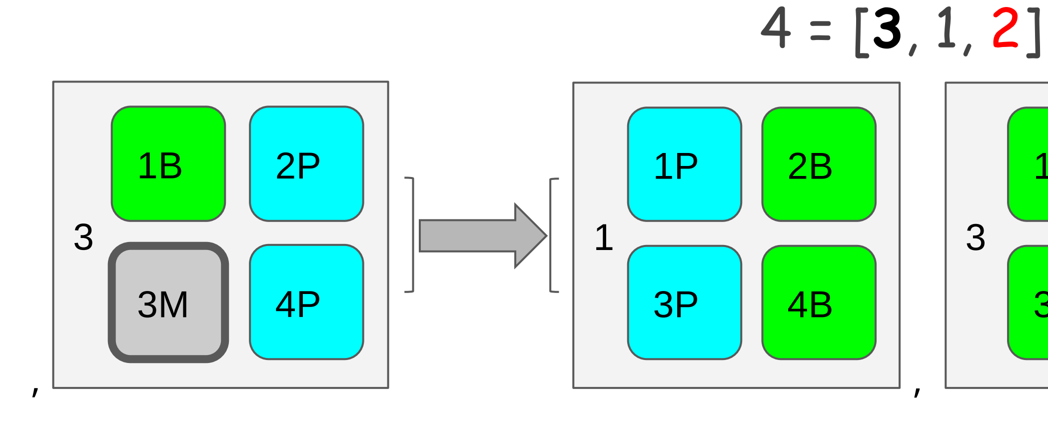
Cluster {
 baseline = true
}



Switch №1 — Rebalancing

Fixed Affinity
1 = [2, **1**, 3],
2 = [2, **3**, 1],
3 = [**1**, 2, 3],
4 = [**3**, 1, 2]





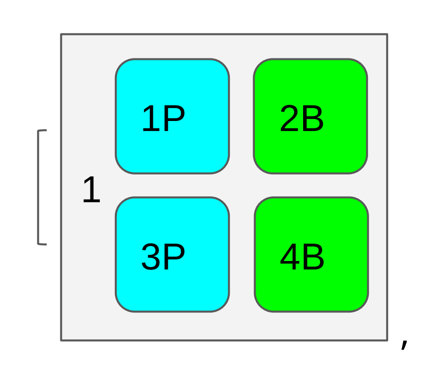
Cache {
partitions = 4,
backups = 2
}

Cluster {
baseline = true
}



Switch №1 — Rebalanced

Fixed Affinity
1 = [2, **1**, 3],
2 = [2, **3**, 1],
3 = [**1**, 2, 3],
4 = [**3**, 1, 2]



3 1B 2P 3B 4P

1 1P 2B 3P 4B

3 1B 2P 3B 4P

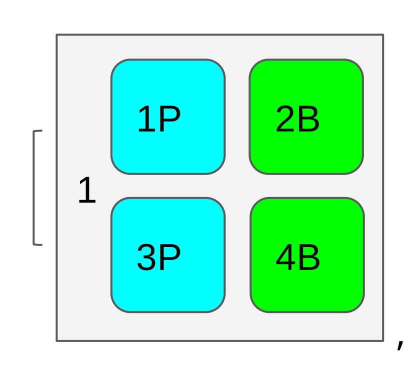
Cache {
partitions = 4,
backups = 2
}

Cluster {
baseline = true
}



Switch №?

Готово за ~~одно~~ два переключения!



3
1B
2P
3B
4P

1
1P
2B
3P
4B

3
1B
2P
3B
4P

```
Cache {
 partitions = 4,
 backups = 2
}

Cluster {
 baseline = true
}
```




Switch №?

Готово за ~~одно~~ два переключения!

[1: 1B, 2M, 3P, 4B ; 2: 1P, 2P, 3B, 4M ; 3: 1B, 2B, 3M, 4P] → [1: 1P, 2B, 3P, 4B ; 3: 1B, 2P, 3B, 4P]

Cache {
partitions = 4,
backups = 2
}

Cluster {
baseline = true
}



Rebalancing vs Rebalanced

[1 (1B, 2M, 3P, 4B), 2 (1P, 2P, 3B, 4M), 3 (1B, 2B, 3M, 4P)] ≠ [1 (1B, 2B, 3P, 4B), 2 (1P, 2P, 3B, 4B), 3 (1B, 2B, 3B, 4P)]



Baseline Node Left on Balanced Cluster

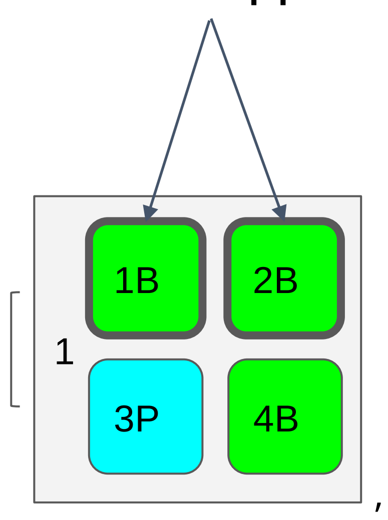
Cache {
partitions = 4,
backups = 2
}

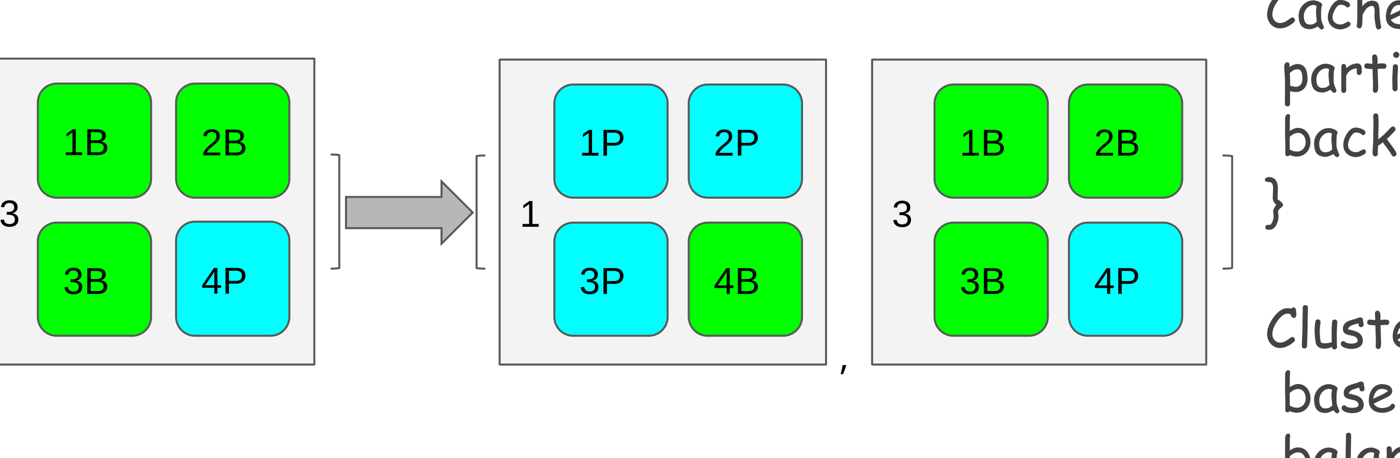
Cluster {
baseline = true,
balanced = true
}



Baseline Node Left on Balanced Cluster

Primary должны быть здесь





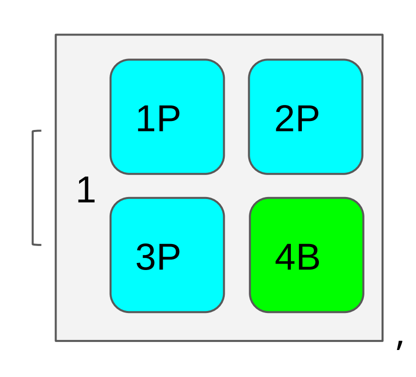

```
Cache {
 partitions = 4,
 backups = 2
}

Cluster {
 baseline = true,
 balanced = true
}
```




Baseline Node Left on Balanced Cluster

Могло быть готово за **0** переключений!



3
1B
2B
3B
4B

1
1P
2P
3P
4B

3
1B
2B
3B
4P

```
Cache {
 partitions = 4,
 backups = 2
}

Cluster {
 baseline = true,
 balanced = true
}
```




Balanced Baseline topology

Отбалансированная Baseline-топология гарантирует, что при выходе узла:

- Партиции останутся на своих местах, не потребуется перемещение партиций (гарантия от *Baseline topology*)
- Не потребуется ребалансировка для переключения *backup-партиций* в *primary* (гарантия от завершенной балансировки)



Промышленная эксплуатация

Отличительные черты “Реального Production”:

- Привязка данных к конкретным физическим узлам
- Отсутствие (завершенность) миграции данных между узлами



PME-free Switch, с версии 2.8

Подготовительные работы

1) Настраиваем *Baseline*
2) Дожидаемся полной ребалансировки кластера

Отказ узла кластера

1) Блокируем новые операции, не блокируя текущие
2) Восстанавливаем данные для вышедших из строя *primary*
3) Переключаем *backup-партиции* в *primary*
4) Разрешаем новые операции



Pme-Free Switch (Блокировки + Recovery)

До версии 2.8 все изменения топологии проходили по единому сценарию

1) Блокируем все новые операции
2) Восстанавливаем операции, “связанные” с вышедшими узлами
3) ~~Дожидаемся завершения уже запущенных операций~~
4) ~~PME (Partition Map Exchange)~~
5) ~~Каждый узел применяет новое распределение~~
6) Начинаем обрабатывать новые операции

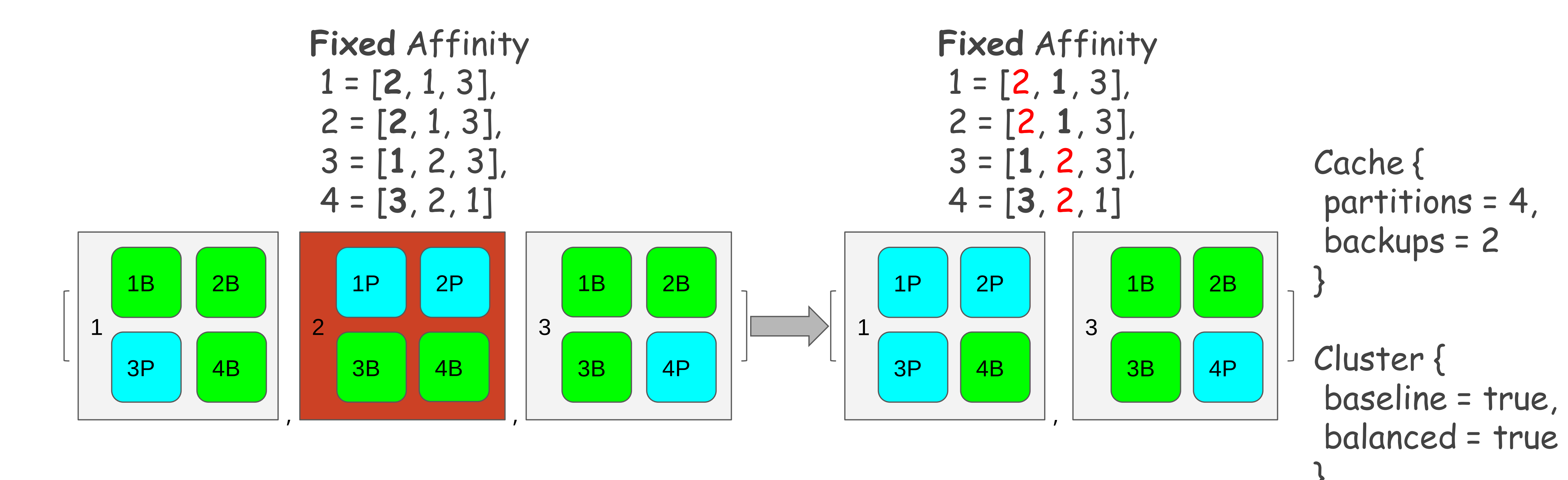


Benchmarking PME vs PME-free switch



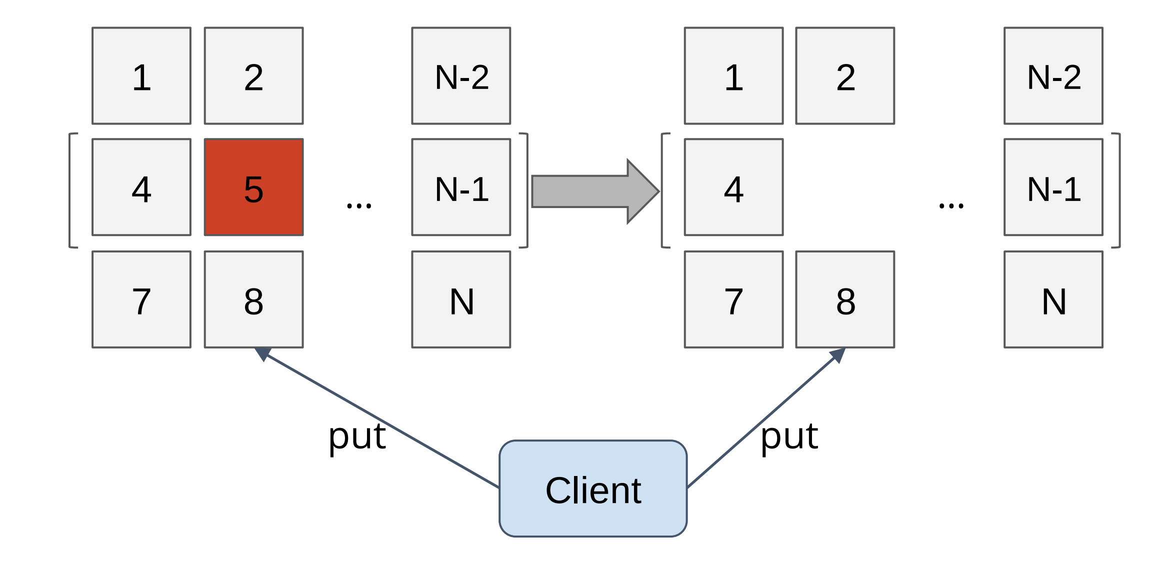


Глобальное vs Локальное переключение



Benchmark: Глобальное vs Локальное переключение

Version	Worst latency (ms)
2.7.6	**192**
2.8.0	**99**



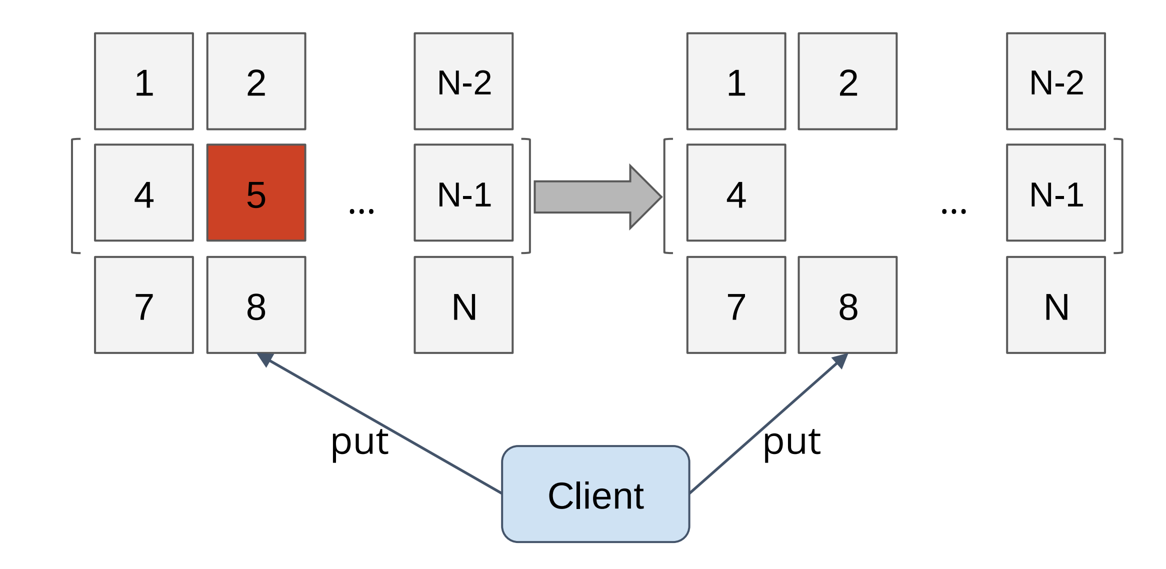

```
Cluster {
 servers = 46,
 clients = 1,
 caches = 1,
 baseline = true,
 balanced = true
}

Server {
 cpu = 8,
 ram = 64gb
}
```




Benchmark: Глобальное vs Локальное переключение

Version	Worst latency (ms)
2.7.6	2396
2.8.0	**536**



Cluster {
servers = 46,
clients = 1,
caches = **100**,
baseline = true,
balanced = true
}

Server {
cpu = 8,
ram = 64gb
}



Отсутствие ожидания запущенных ранее транзакций

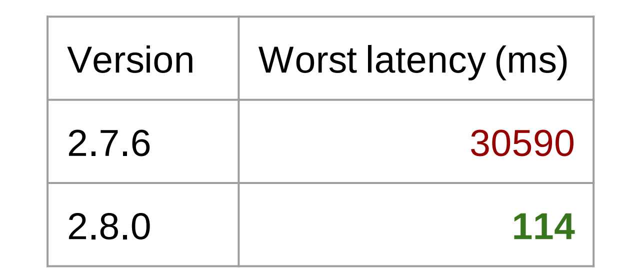
Топология для них останется прежней, но с повреждениями, и транзакции либо “сломаются”, либо продолжат выполняться на *урезанной топологии*.


```
Cache {
 partitions = 4,
 backups = 2
}

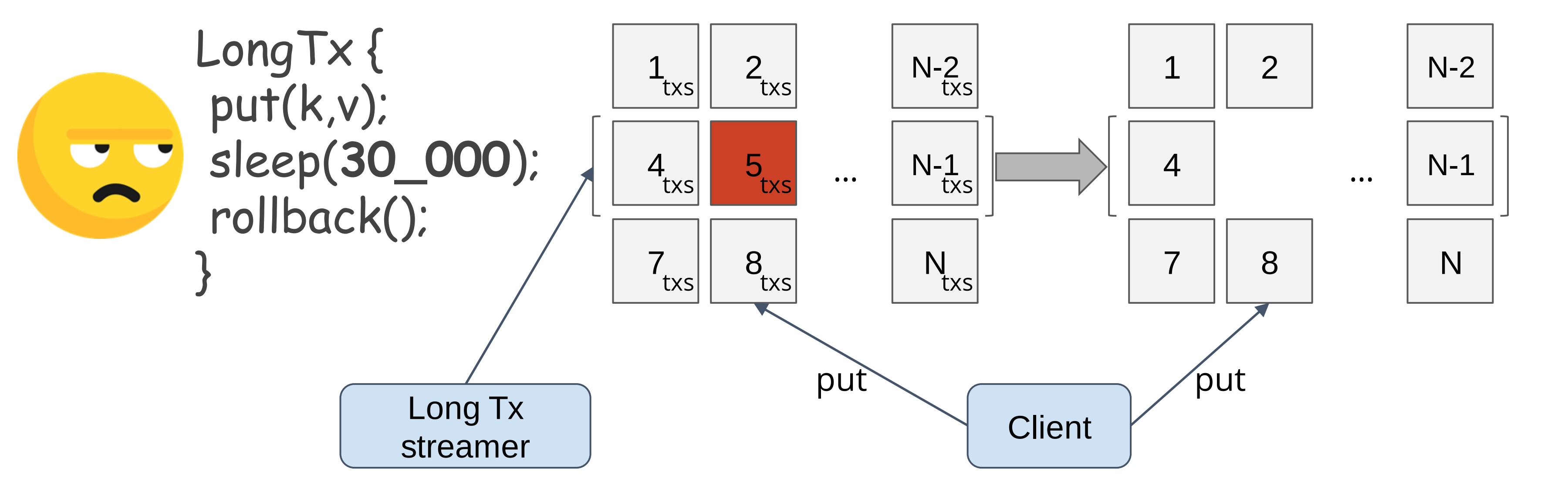
Cluster {
 baseline = true,
 balanced = true
}
```




Benchmark: отсутствие ожидания запущенных ранее транзакций



Version	Worst latency (ms)
2.7.6	30590
2.8.0	114




```
Cluster {
 servers = 46,
 clients = 1,
 caches = 1,
 baseline = true,
 balanced = true
}

Server {
 cpu = 8,
 ram = 64gb
}
```




Pme-Free Switch (Блокировки + Recovery)

До версии 2.8 все изменения топологии проходили по единому сценарию

1) Блокируем все новые операции
2) Восстанавливаем операции, “связанные” с вышедшими узлами
3) ~~Дожидаемся завершения уже запущенных операций~~
4) ~~PME (Partition Map Exchange)~~
5) ~~Каждый узел применяет новое распределение~~
6) Начинаем обрабатывать новые операции



Pme-Free Switch (Блокировки + Recovery)

До версии 2.8 все изменения топологии проходили по единому сценарию

1) Блокируем все новые операции
2) Восстанавливаем операции, “связанные” с вышедшими узлами
3) ~~Дожидаемся завершения уже запущенных операций~~
4) ~~PME (Partition Map Exchange)~~
5) ~~Каждый узел применяет новое распределение~~
6) Начинаем обрабатывать новые операции



Tx Recovery

Транзакции требуют восстановления, если вышел узел с *primary-партицией* транзакции.

Primary

1 | 1B | 2B tx | 3P | 4B
2 | 1P | 2P tx | 3B | 4B
3 | 1B | 2B tx | 3B | 4P

Backups

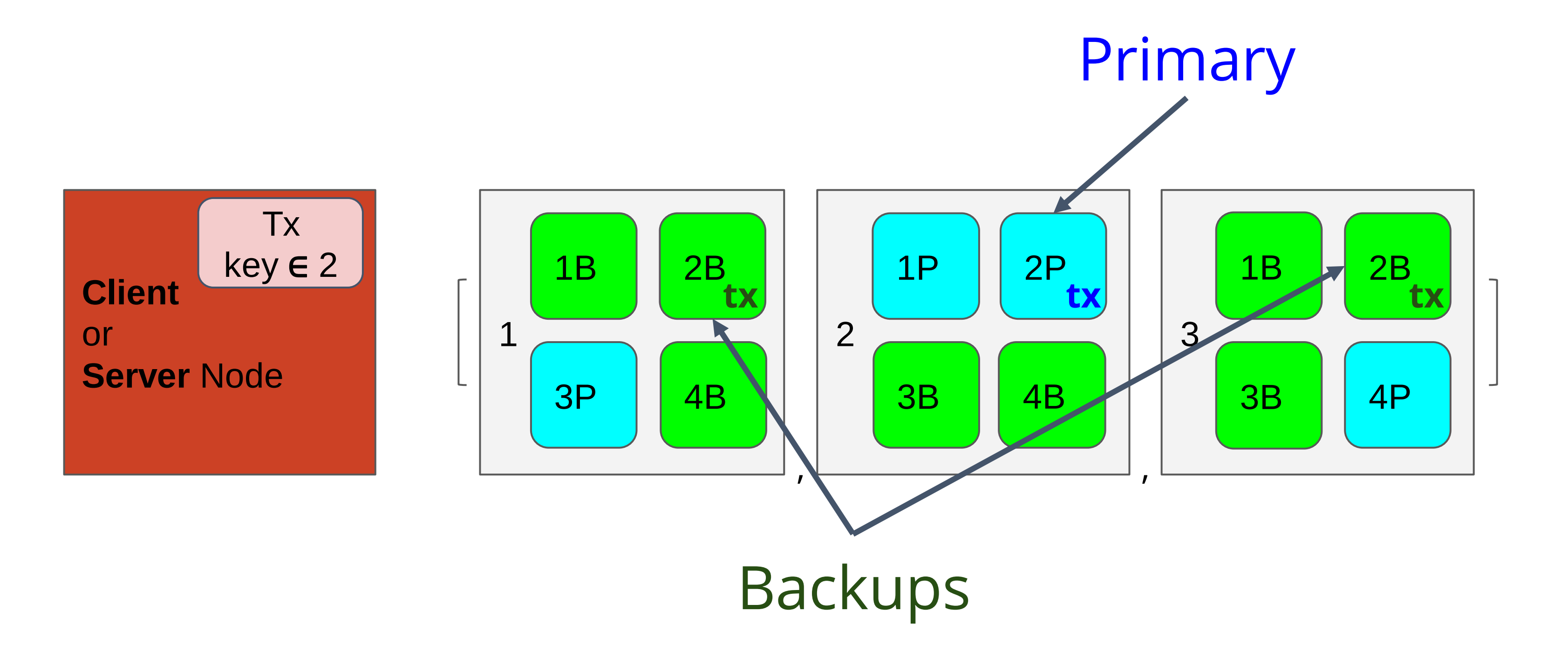
```
Cache {
 partitions = 4,
 backups = 2
}

Cluster {
 baseline = true,
 balanced = true
}
```




Tx Recovery

Транзакции требуют восстановления, если вышел узел-координатор (инициировавший транзакцию)




```
Cache {
 partitions = 4,
 backups = 2
}

Cluster {
 baseline = true,
 balanced = true
}
```




One Phase Commit (1PC)

Keys 1, 4, and 7 map to the same partition

Client

Primary Node

Backup Node

start transaction

lock key 1

lock key 4

lock key 7

commit backups for keys 1, 4, 7

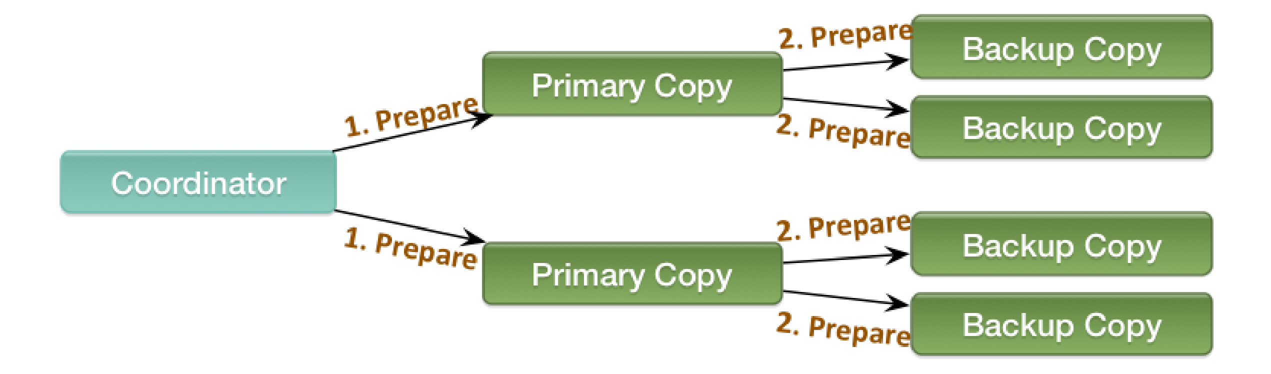
acknowledge commit

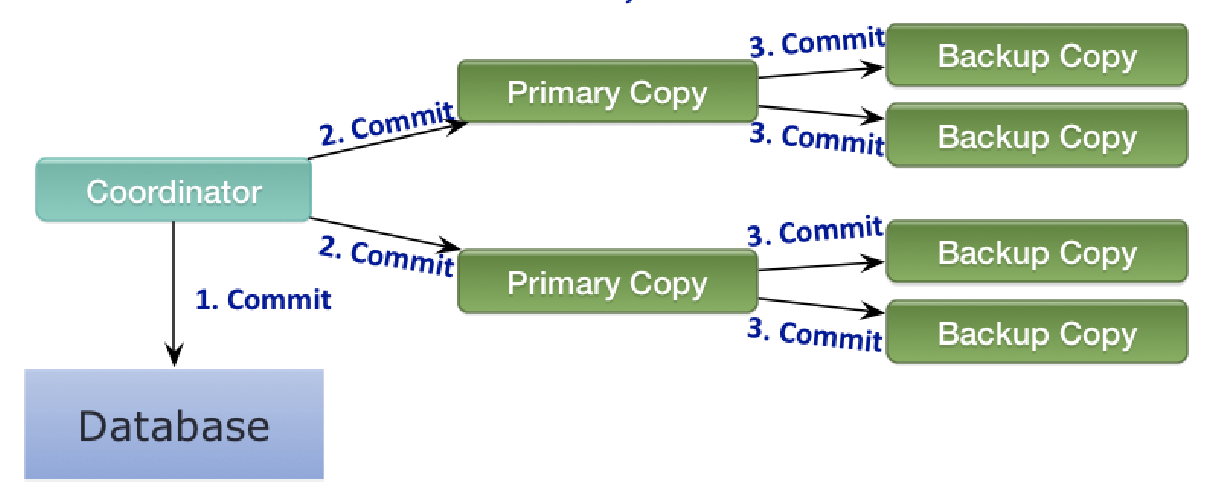
commit primary copies

acknowledge



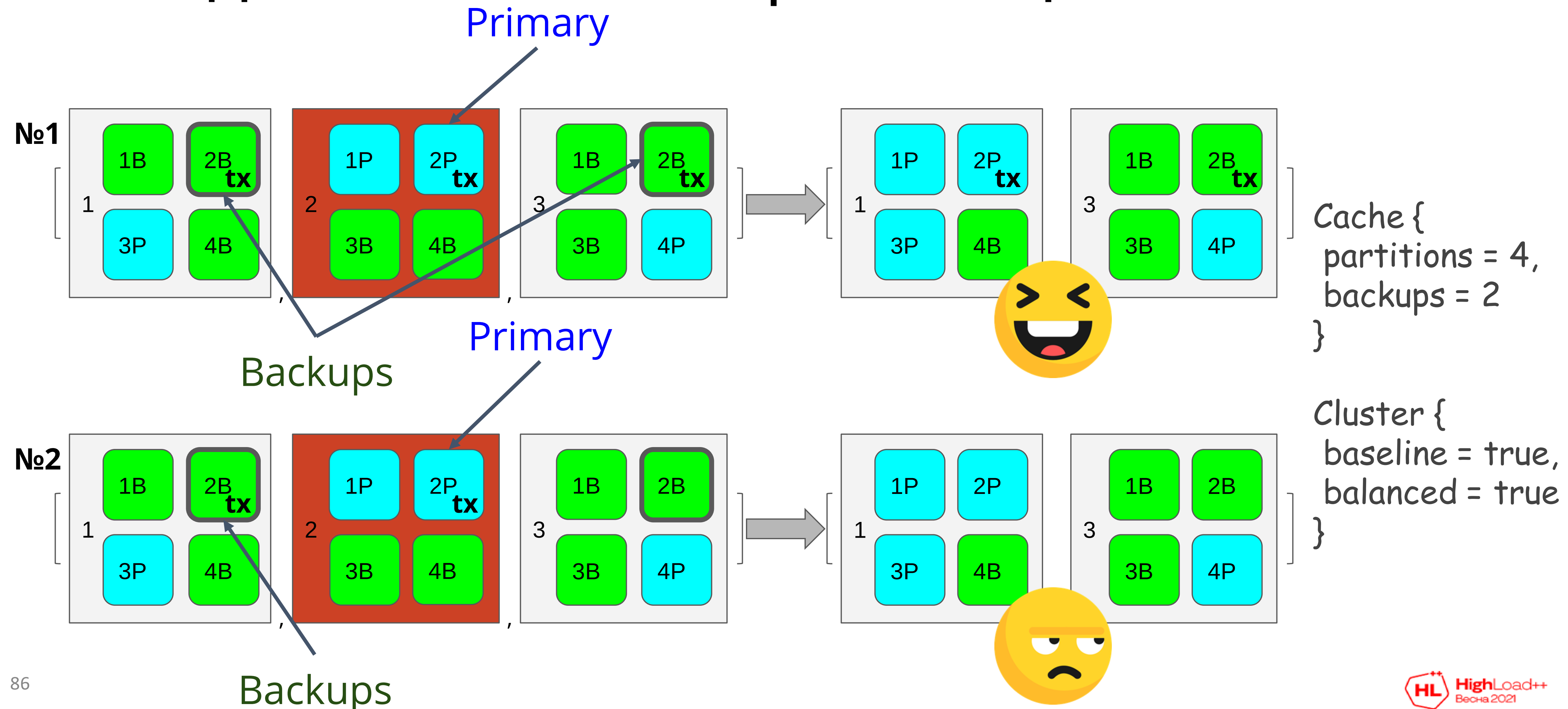
Two-Phase Commit (2PC)







Подготовленные транзакции



А можно восстановиться еще быстрее?





Уменьшение зоны восстановления

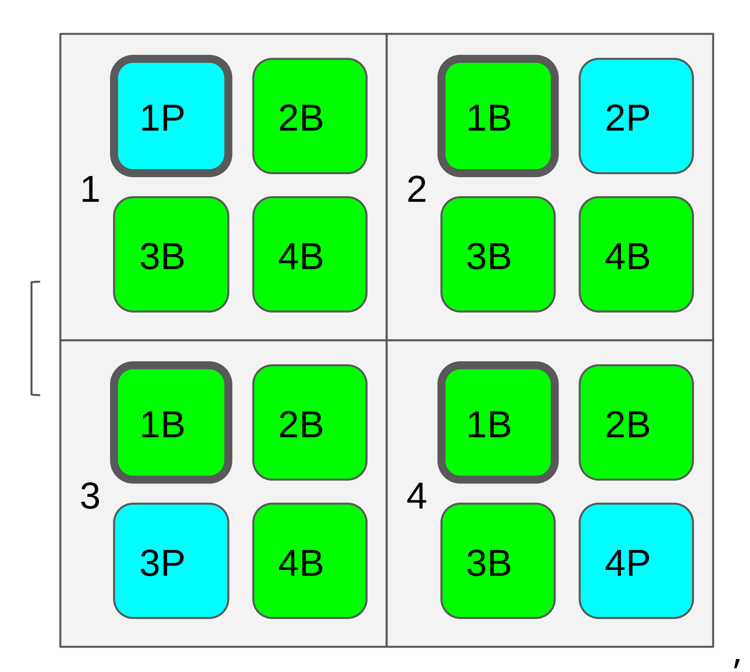
Узел кластера, не содержащий *backup-партиций*, связанных с вышедшими *primary*, не требует восстановления.

Необходимо уменьшить список узлов содержащих *backup-партиции* для *primary-партиций* каждого из узлов.





Cellular affinity



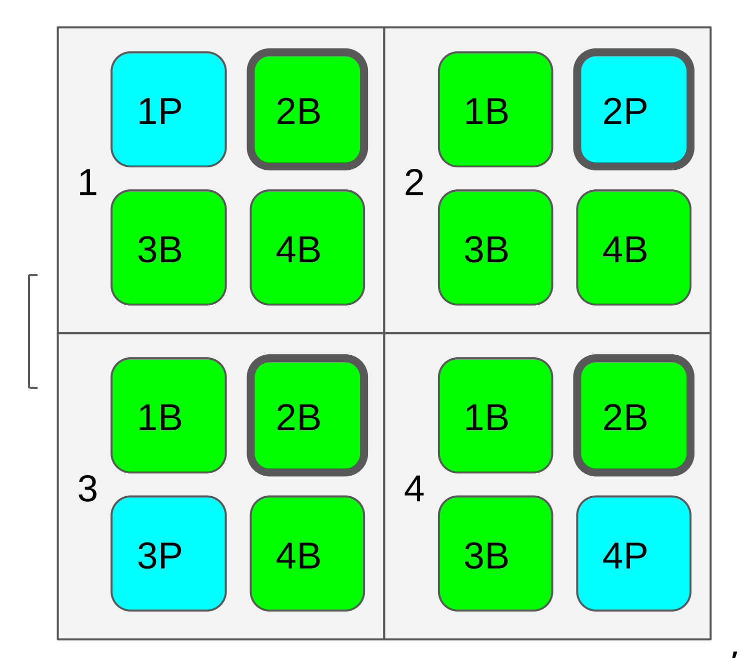
5
5P
6B
7B
8B
6
5B
6P
7B
8B
7
5B
6B
7P
8B
8
5B
6B
7B
8P

Cache {
partitions = 8,
backups = 3,
cell-affinity = **true**
}

Cluster {
baseline = true,
balanced = true
}

Cell size = backups +1

Cellular affinity



5
5P 6B
7B 8B

6
5B 6P
7B 8B

7
5B 6B
7P 8B

8
5B 6B
7B 8P

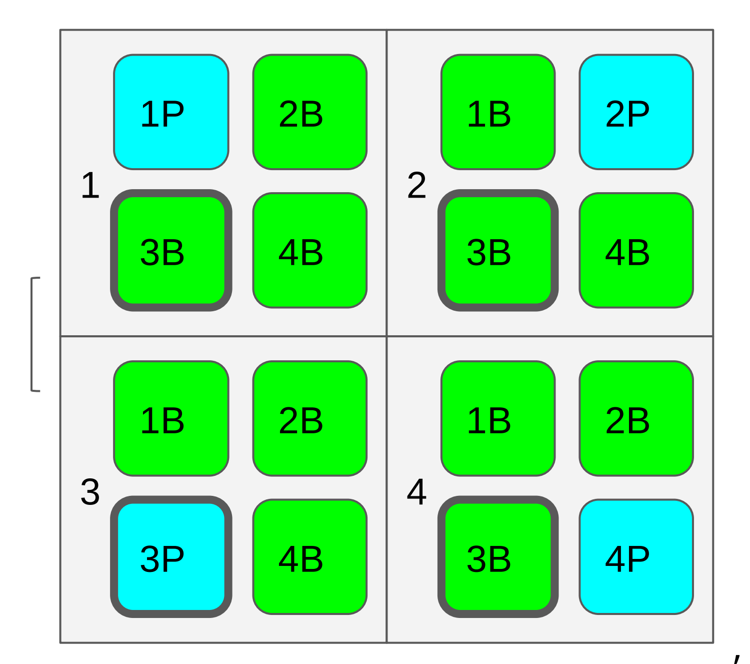
```
Cache {
 partitions = 8,
 backups = 3,
 cell-affinity = true
}

Cluster {
 baseline = true,
 balanced = true
}
```

Cell size = backups +1



Cellular affinity



5
5P
6B
7B
8B
6
5B
6P
7B
8B
7
5B
6B
7P
8B
8
5B
6B
7B
8P

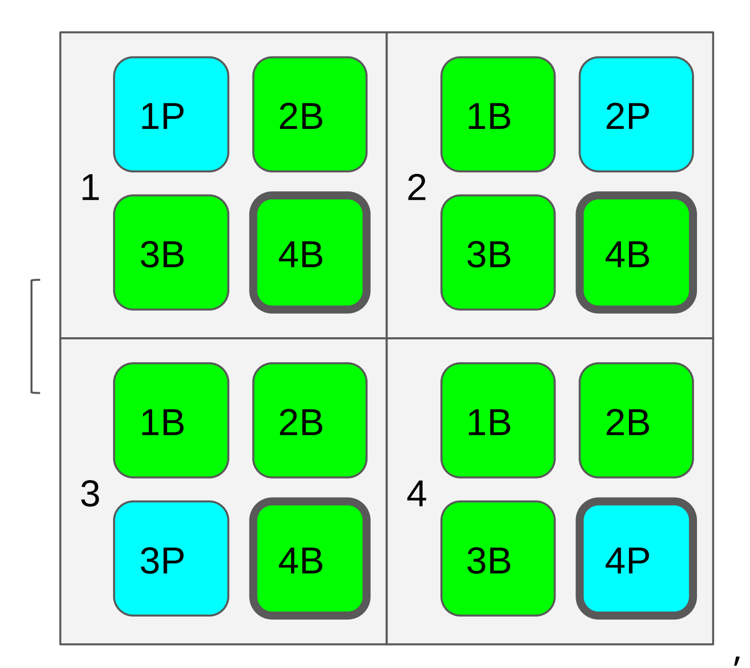
```
Cache {
 partitions = 8,
 backups = 3,
 cell-affinity = true
}

Cluster {
 baseline = true,
 balanced = true
}
```

Cell size = backups +1



Cellular affinity



5
5P 6B
7B 8B

6
5B 6P
7B 8B

7
5B 6B
7P 8B

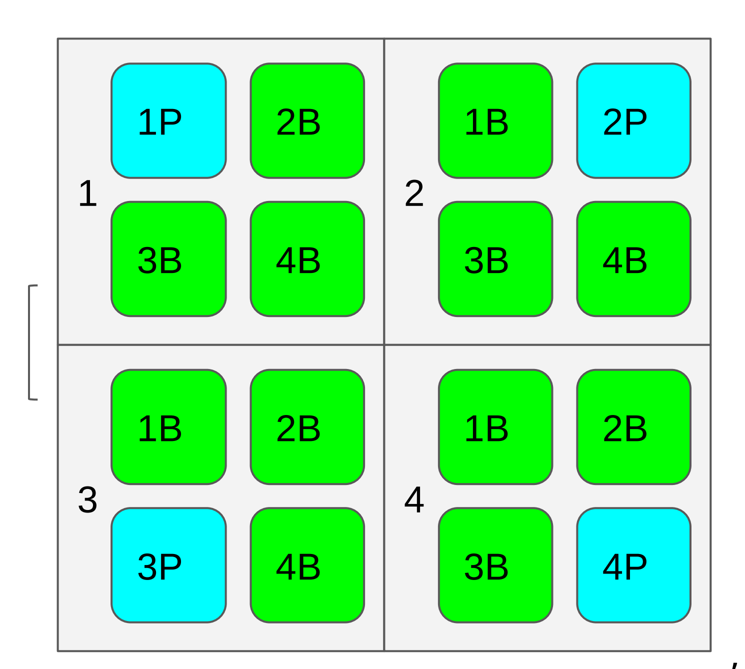
8
5B 6B
7B 8P

Cache {
partitions = 8,
backups = 3,
cell-affinity = **true**
}

Cluster {
baseline = true,
balanced = true
}

Cell size = backups +1








```
Cache {
 partitions = 8,
 backups = 3,
 cell-affinity = true
}

Cluster {
 baseline = true,
 balanced = true
}
```

Cell size = backups +1

1	1P	2B	2	1B	2P	5	5P	6B	6	5B	6P
	3B	4B		3B	4B		7B	8B		7B	8B
3	1B	2B	4	1B	2B	7	5B	6B	8	5B	6B
	3P	4B		3B	4P		7P	8B		7B	8P

2	1B	2P
	3B	4B

Cache {
partitions = 8,
backups = 3,
cell-affinity = true
}

Cluster {
baseline = true,
balanced = true
}

Cell size = backups +1



1: 1P 2B 3B 4B
2: 1B 2P 3B 4B
3: 1B 2B 3P 4B
4: 1B 2B 3B 4P
5: 5P 6B 7B 8B
6: 5B 6P 7B 8B
7: 5B 6B 7P 8B
8: 5B 6B 7B 8P

− 2: 1B 2P 3B 4B

1: 1P 2B 3B 4B
3: 1B 2B 3P 4B
4: 1B 2B 3B 4P
5: 5P 6B 7B 8B
6: 5B 6P 7B 8B
7: 5B 6B 7P 8B
8: 5B 6B 7B 8P

```
Cache {
 partitions = 8,
 backups = 3,
 cell-affinity = true
}

Cluster {
 baseline = true,
 balanced = true
}
```

Cell size = backups +1



Cellular affinity — Partial switch

1	1P	2B	
	3B	4B	
3	1B	2B	
	3P	4B	
4	1B	2B	
	3B	4P	
5	5P	6B	
	7B	8B	
6	5B	6P	
	7B	8B	
7	5B	6B	
	7P	8B	
8	5B	6B	
	7B	8P	

Switch in progress

Switch finished

```
Cache {
 partitions = 8,
 backups = 3,
 cell-affinity = true
}

Cluster {
 baseline = true,
 balanced = true
}
```

Cell size = backups +1



Cellular affinity — Switch complete

1 | 1P | 2P | 3B | 4B

3 | 1B | 2B | 3P | 4B

4 | 1B | 2B | 3B | 4P

5 | 5P | 6B | 7B | 8B

6 | 5B | 6P | 7B | 8B

7 | 5B | 6B | 7P | 8B

8 | 5B | 6B | 7B | 8P

Switch finished

Switch finished

Cache {
partitions = 8,
backups = 3,
cell-affinity = true
}

Cluster {
baseline = true,
balanced = true
}

Cell size = backups +1



Benchmarking PME-free vs Cellular switch





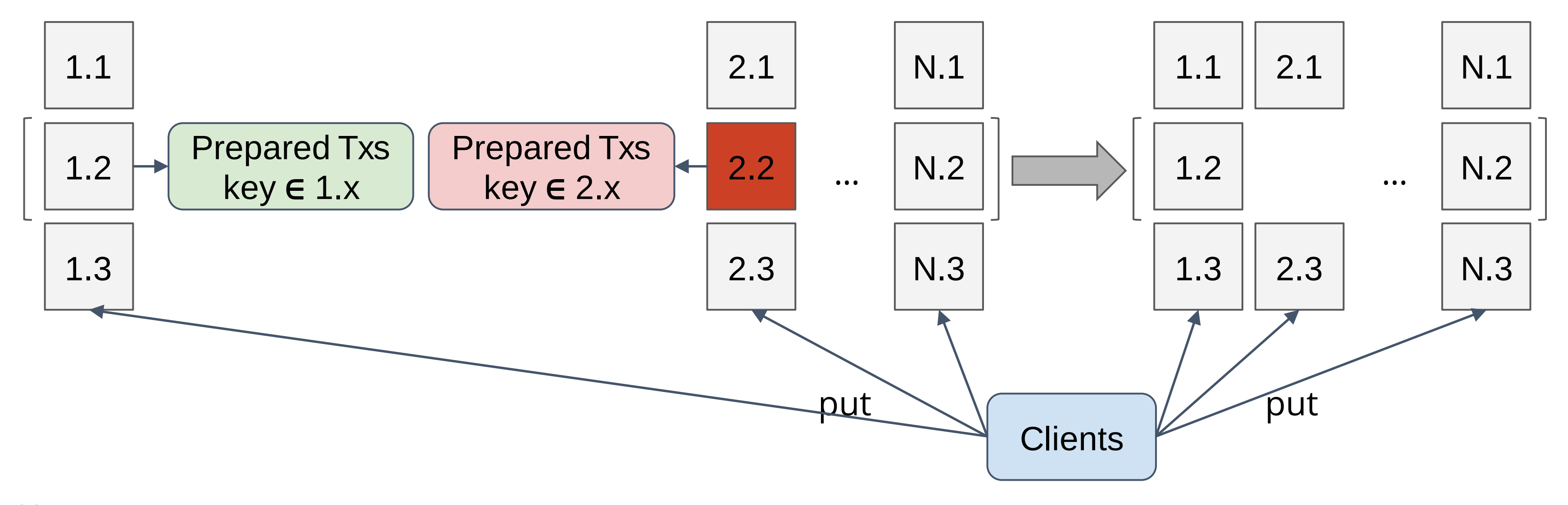
Benchmark: Cellular switch #colocated

Version	Worst latency (ms)
2.8.1	alive cells ≈ 506 .. 537 broken cell = 500
2.11.0*	alive cells ≈ **22 .. 64** broken cell = 572

```
PreparedTxs {
 amount = 500 // per cell
}
```

```
Cluster {
 servers = 33, // 11x3
 clients = 11,
 caches = 1,
 baseline = true,
 balanced = true,
 cell-affinity = true
}
```

```
Server {
 cpu = 8,
 ram = 64gb
}
```





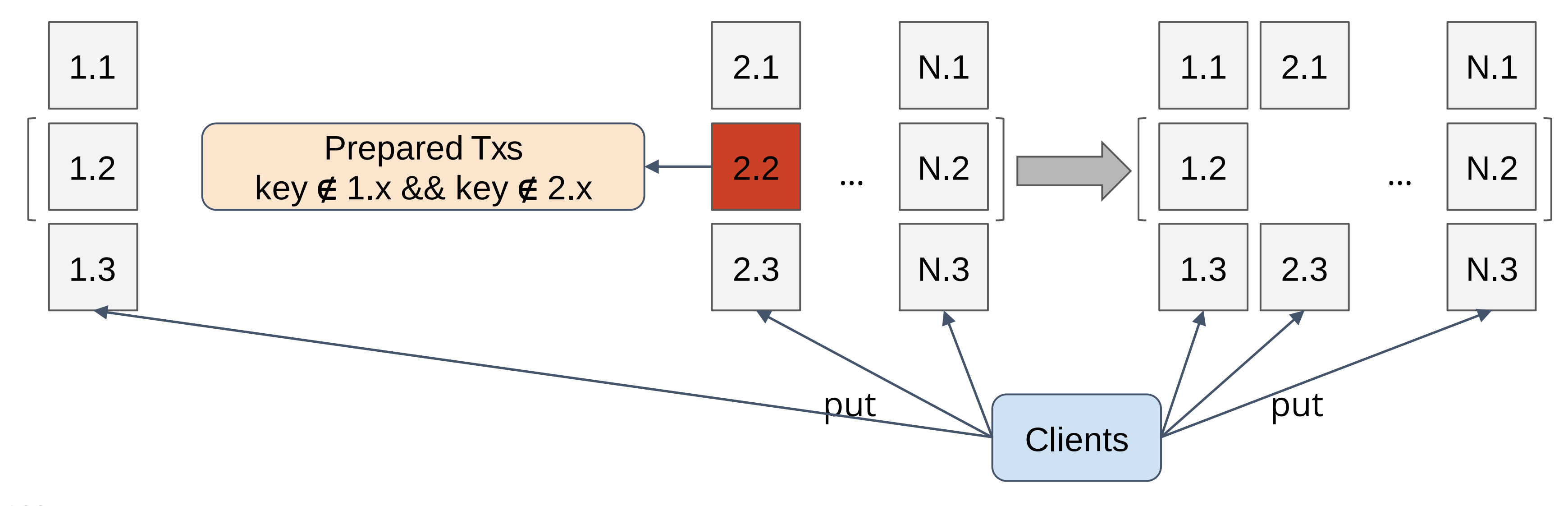
Benchmark: Cellular switch #non-colocated

Version	Worst latency (ms)
2.8.1	non-affected alive cell = 426 affected alive cells ≈ 406 .. 438 broken cell = 394
2.11.0*	non-affected alive cell = **41** affected alive cells ≈ **45 .. 81** broken cell = **47**

PreparedTxs {
 amount ≈ **500** // per cell
}

Cluster {
 servers = 33, // 11x3
 clients = 11,
 caches = 1,
 baseline = true,
 balanced = true,
 cell-affinity = true
}

Server {
 cpu = 8,
 ram = 64gb
}





Cellular switch : нюансы #1

К сожалению, наличие *replicated-кэшей* гарантирует, что каждый узел кластера будет содержать *backup-партиции* для вышедших *primary*.

Т.е. для *replicated-кэшей* ничего не ускорилось.

Но справочники обновляются редко.

Ожидаем быстрый/мгновенный глобальный Recovery по *replicated-кэшам*.



Cellular switch : нюансы #2

Транзакции могут оперировать ключами из разных ячеек.

```
try (Transaction tx = ignite.transactions().txStart()) {
    cache.put(k1, v1); // key from broken cell
    cache.put(k2, v2); // key from alive cell
    cache.put(k3, v3); // key from another alive cell
    ...
```

Неповрежденные ячейки будут ждать *recovery* таких транзакций.

Но не полного восстановления поврежденных ячеек.



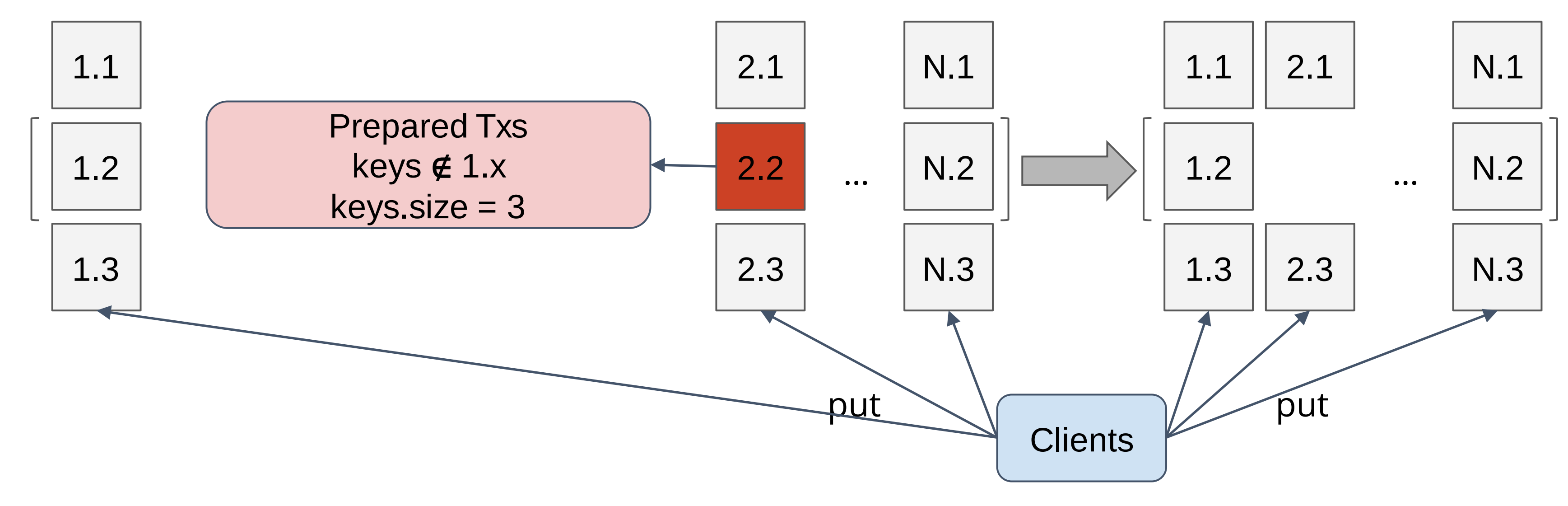
Benchmark: Cellular switch #multi-key

PreparedTxs {
 amount ≈ **500** // per cell
}

Version	Worst latency (ms)
2.8.1	non-affected alive cell = 1769 affected alive cells ≈ 1738 .. 1762 broken cell = 1827
2.11.0*	non-affected alive cell = **69** affected alive cells ≈ **1029 .. 1419** broken cell = 1885

Cluster {
 servers = 33, // 11x3
 clients = 11,
 caches = 1,
 baseline = true,
 balanced = true,
 cell-affinity = true
}

Server {
 cpu = 8,
 ram = 64gb
}





Cellular Switch (для неповрежденных ячеек)

До версии 2.8 все изменения топологии проходили по единому сценарию

1) Блокируем все новые операции
2) ~~Восстанавливаем операции “связанные” с вышедшими узлами~~
3) ~~Дожидаемся завершения уже запущенных операций~~
4) ~~PME (Partition Map Exchange)~~
5) ~~Каждый узел применяет новое распределение~~
6) Начинаем обрабатывать новые операции

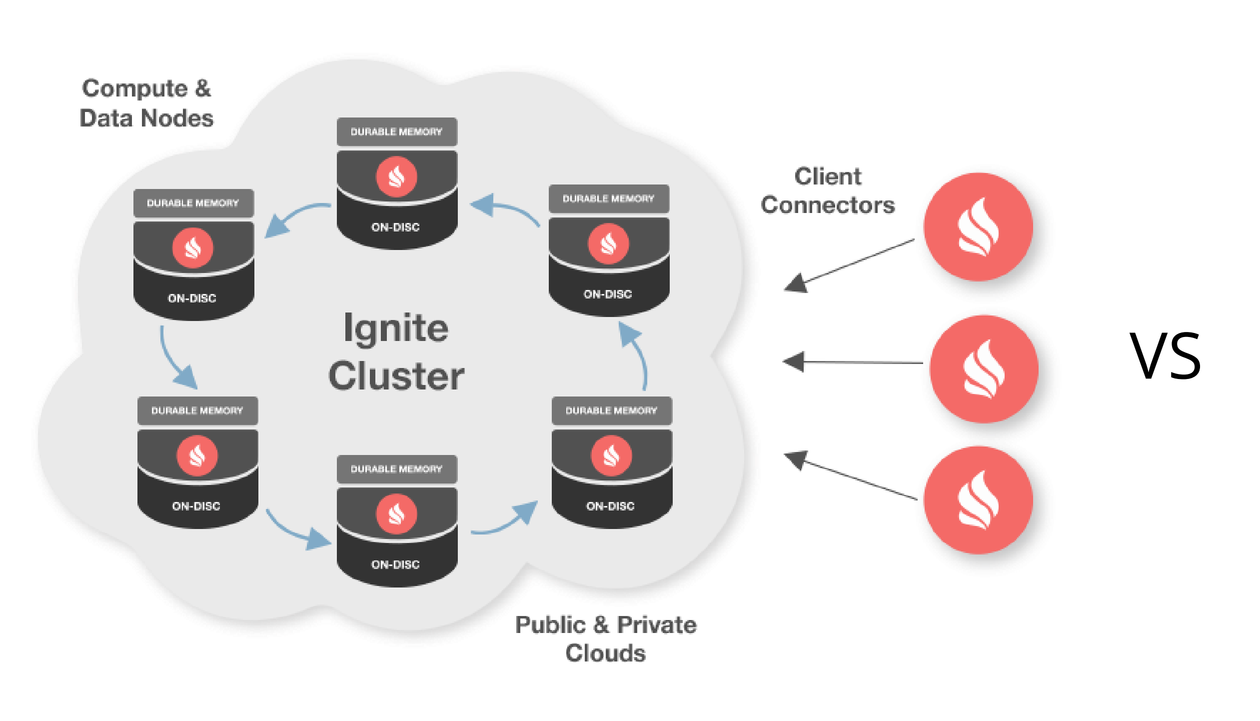


Однако, всё не так просто



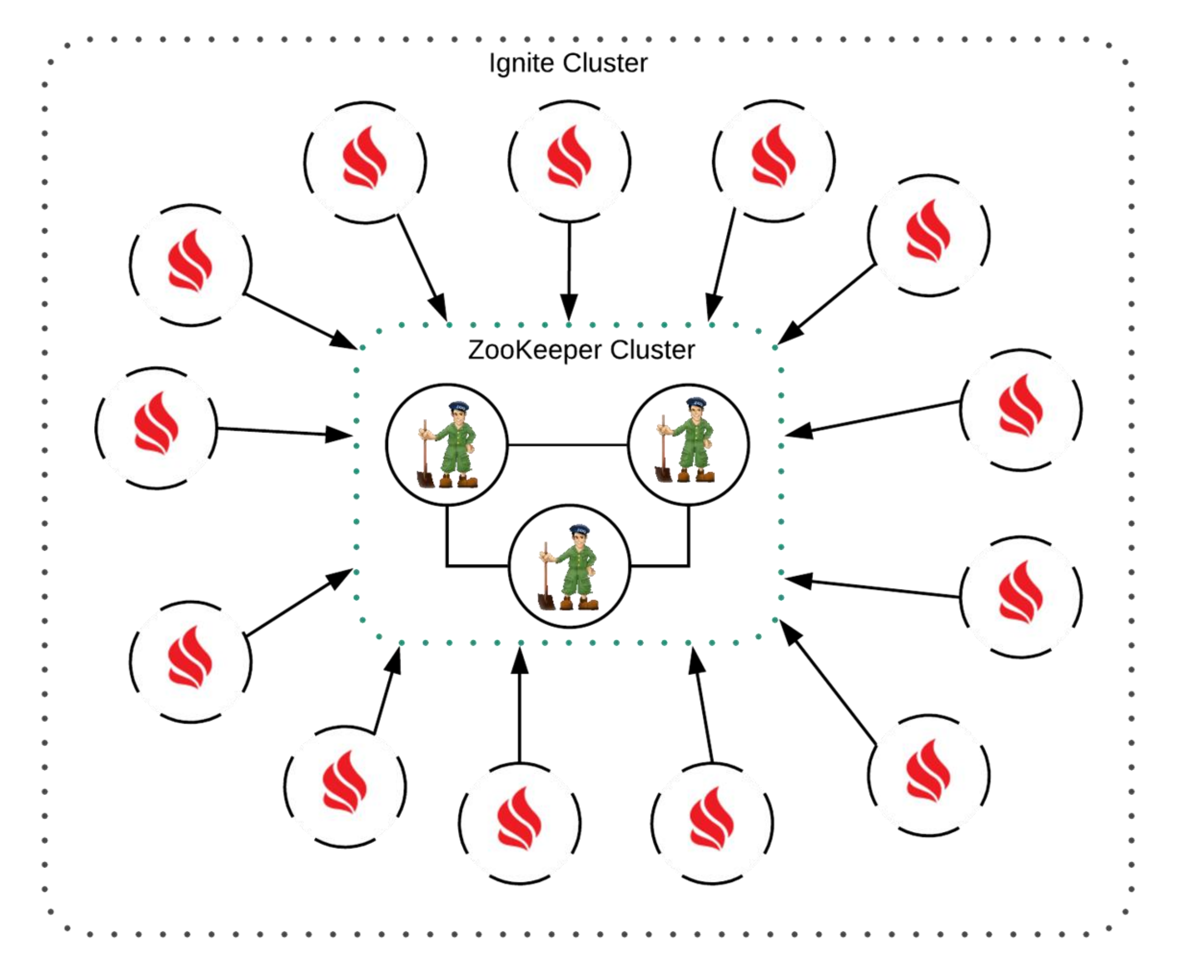


Discovery (Ring vs Star)



TcpDiscovery

VS



ZooKeeperDiscovery



Cluster.Size--



Node left

- Плановый вывод из топологии в *Shutdown Hook* (***SIGTERM***)
- Выход при **критической ошибке** при *<u>правильно</u>* сконфигурированном *FailureHandler*

```java
/**
 * Handler will stop node in case of critical error using {@code IgnitionEx.stop(nodeName, true, true)} call.
 */
public class StopNodeFailureHandler extends AbstractFailureHandler {
    /** {@inheritDoc} */
    @Override public boolean handle(Ignite ignite, FailureContext failureCtx) {
        new Thread(
            new Runnable() {
                @Override public void run() {
                    U.error(ignite.log(), msg: "Stopping local node on Ignite failure: [failureCtx=" + failureCtx + ']');

                    IgnitionEx.stop(ignite.name(), cancel: true, shutdown: null, stopNotStarted: true);
                }
            },
            name: "node-stopper"
        ).start();

        return true;
    }
```

оповещают кластер о выходе узла.



Node failed

- **SIGKILL**
- **JVM** crash
- **OS** crash
- **Hardware** crash

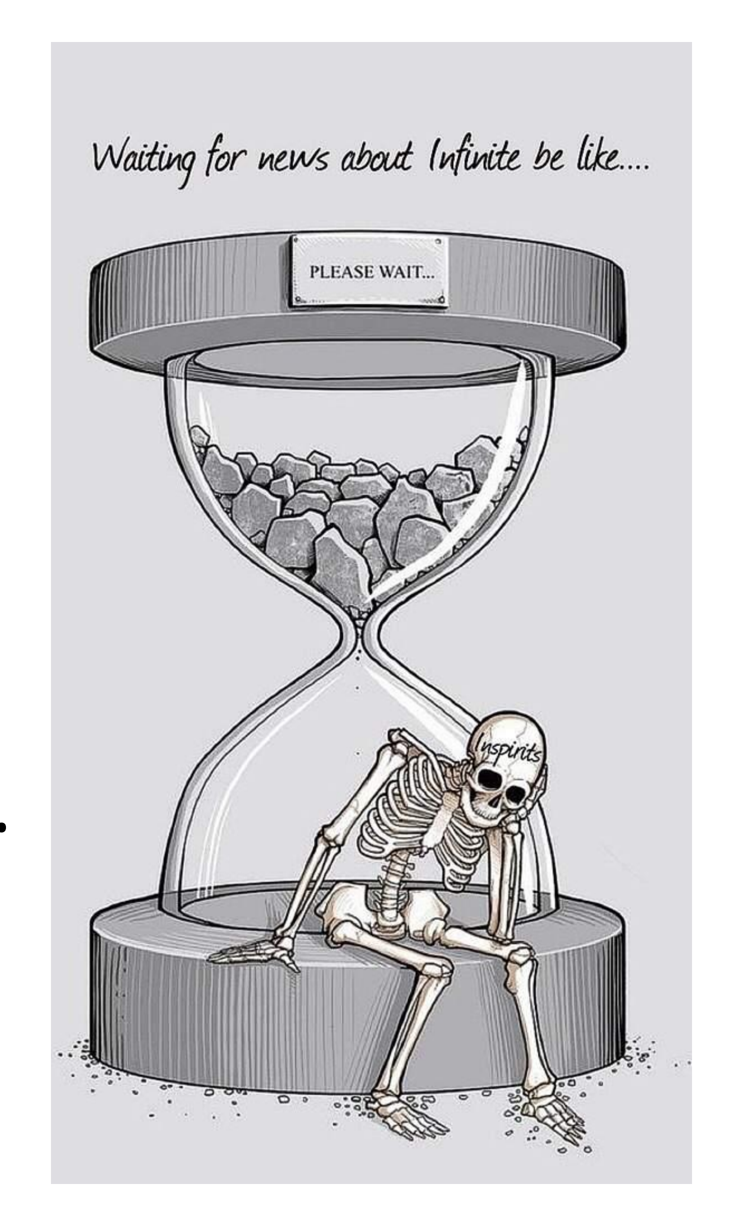
DON'T SIGKILL
USE SIGTERM
JAVA
MYSQL
XORG
APACHE
PHP
THINK ABOUT THE THREADS



Timed out

- Долгий **GC**
- **CPU** usage is at 100%
- Broken **network**

Выводим узел из топологии по факту истечения допустимого времени ожидания.





Txs recovery time ≈ **400** ms,
Failure detection timeout = **2** sec

Cellular switch - Worst latency (ms)

	Left (SIGTERM, FailureHandler)	**Failed** (SIGKILL, JVM/OS/HW Crash)	**Timed out** (GC, Broken network, CPU 100%)
TcpDiscovery (Ring)	alive cells ≈ **44 .. 71** broken cell = **410**	alive cells ≈ **31 .. 69** broken cell = **419**	alive cells ≈ **33 .. 87** broken cell = **1932**
ZooKeeper	alive cells ≈ **29 .. 40** broken cell = **427**	alive cells ≈ **24 .. 46** broken cell = **2571**	alive cells ≈ **32 .. 40** broken cell = **2967**

Настройки vs Оборудование

ZooKeeper config:

tickTime=166

minSessionTimeout=500

maxSessionTimeout=500

==========>

WARN [SyncThread:1:FileTxnLog@343] - fsync-ing the write ahead log in SyncThread:1 took 1731ms which will adversely effect operation latency. File size is 67108880 bytes. See the ZooKeeper troubleshooting guide



Итого (Ignite best practices)

1) **Tcp**Discovery < **20 .. 40** узлов < **Zookeeper**Discovery.
2) **SIGTERM** для вывода узла.
3) Правильный **FailureHandler** — ускоряет crash recovery.
4) **Defaults** — работают везде, но **неоптимально**.
5) **Оборудование** должно соответствовать **требованиям**.
6) Ваша нагрузка **уникальна**, на нее нужны **профильные бенчмарки** (и на crash recovery в том числе!).



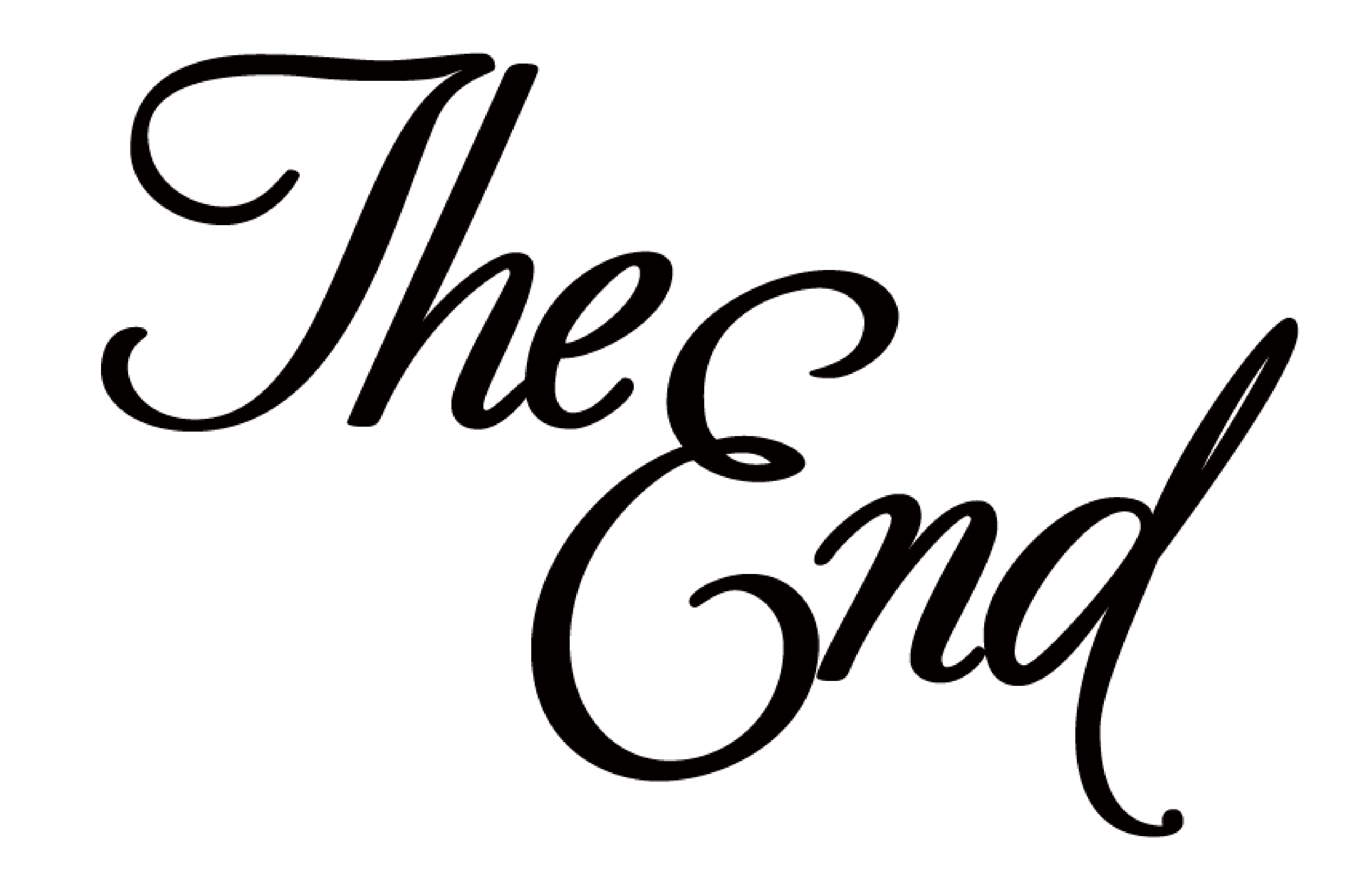


av@apache.org

ignite-summit.org, May 25, 2021, Online, Free

github.com/apache/ignite/tree/ignite-ducktape/modules/ducktests




The End


HL
HighLoad++
Весна 2021